BAB 6

INSTRUKSI MIKROPROSESOR

Setiap mikroprosesor selalu dirancang dan dilengkapi dengan perangkat instruksi. Bentuk perangkat instruksi masing-masing mikroprosesor bergantung jenis arsitektur yang digunakan. Seperti telah dibahas dalam Bab 2 arsitektur mikroprosesor ada tiga jenis yaitu CISC, RIS, dan Super Skalar. Dalam Bab 6 ini diuraikan arsitektur software instruksi mikroprosesor Zilog Z-80 CPU.
Kata kunci: *set instruksi, CISC*

Mikroprosesor Zilog Z-80 CPU adalah salah satu jenis mikroprosesor yang menggunakan arsitektur CISC. Jumlah instruksi Z-80 CPU cukup banyak yaitu sekitar 141 jenis. Instruksi Z-80 CPU dapat digolongkan menjadi 13 kelompok yaitu:

- Instruksi Transfer Data 8 bit
- Instruksi Transfer Data 16 bit
- Instruksi Pertukaran Data
- Instruksi Pelacakan/Search Data
- Instruksi Aritmetika dan Logika 8 bit
- Instruksi Aritmetika Tujuan Umum dan Kendali CPU
- Instruksi Aritmetika 16 bit
- Instruksi Putar dan Geser
- Instruksi Manipulasi bit
- Instruksi Jump
- Instruksi Call dan Return
- Instruksi RESTART
- Instruksi Input dan Output

1. Instruksi Transfer Data

Operasi transfer data atau lebih tepat disebut sebagai operasi copy data pada mikroprosesor Z-80 CPU sebagian besar dijalankan menggunakan perintah LD singkatan dari LOAD. Z-80 CPU memiliki 134 perintah LOAD. Disamping itu ada 6 jenis perintah EXCHANGE yang disingkat dengan EX, EXX. Mikroprosesor Z-80 CPU juga memiliki 12 jenis perintah PUSH dan POP yang digunakan untuk transfer data dalam operasi stack.

Data dapat ditransfer dalam 8 bit atau 16 bit. Perintah transfer data memuat dua operand yaitu operand pertama menunjukkan lokasi dimana data akan disimpan, apakah dalam register atau di memori. Operand pertama Ini disebut **Destinasi.** Operand yang kedua menunjukkan lokasi asli atau asal sebuah data. Operand kedua ini disebut **Source**. Operand dapat berupa register, memori, atau data immediate. Lebar data yang ditransfer dapat berupa data 8 bit atau data 16 bit. Bentuk umum transfer data pada Z-80 CPU adalah seperti Gambar 6.1.

LD (operand destinasi), (operand Source)

Gambar 6.1. Bentuk umum perintah transfer data

Sebagai contoh instruksi LD A, B menunjukkan perintah untuk meng-copy data yang ada di Register B ke Register A. Dalam hal ini Register A berfungsi sebagai destinasi dan Register B berfungsi sebagai Source atau asal/sumber data.

1.1. Transfer Data 8 Bit

Instruksi transfer data 8 bit dapat terjadi diantara:

- Register Ke Register
- Memori Ke Register
- Data Immediate Ke Register
- Register Ke Memori
- Memori Ke Memori
- Data Immediate Ke Memori

1.1.1. Transfer data 8 bit dari Register ke Register

Transfer data 8 bit dari register ke register adalah proses meng-copy data dari sumber atau source data ke tujuan tempat simpan data yang baru. Keduanya baik source dan destinasi adalah register 8 bit. Dalam kasus ini ukuran data yang di-copy-kan adalah 8 bit dan register bekerja baik sebagai sumber dan tujuan sama-sama register 8 bit. Transfer data 8 bit dari register ke register dapat terjadi diantara register 8 bit yaitu register A,B,C,D,E,H,L, dan I. Tranfer data itu bisa terjadi diantara register A dengan register B, diantara register A dengan register C dan sebagainya. Untuk lebih memahami proses transfer data 8 bit dari register ke register perhatikan contoh kasus Gambar 6.2.

No	Assembly	Operasi	Keterangan
1.	**LD A,B**	A ← B	isi register A dengan data dari register B
2.	**LD B,C**	B ← C	isi register B dengan data dari register C
3.	**LD B,A**	B ← A	isi register B dengan data dari register A
4.	**LD B,E**	B ← E	isi register B dengan data dari register E

Gambar 6.2. Contoh-contoh transfer data 8 bit register ke register

Contoh kasus nomor 1 Gambar 6.2 menunjukkan transfer data 8 bit dari register B ke register A. Data yang ada di register B di-copy-kan ker egister A. Dalam kasus ini register B sebagai source dan register A sebagai destinasi. Pada kasus ke 2 transfer data 8 bit terjadi dari register C ke register B. Data yang ada di register C di-copy-kan ke register B. Dalam kasus ini register C sebagai source dan register B sebagai destinasi. Pada kasus ke 3 transfer data 8 bit terjadi dari register A ke register B. Data yang ada di register A di-copy-kan ke register B. Dalam kasus ini register A sebagai source dan register B sebagai destinasi. Pada kasus ke 4 transfer data 8 bit terjadi dari register E ke register B. Data yang ada di register E di-copy-kan ke register B. Dalam kasus ini register E sebagai source dan register B sebagai destinasi. Bentuk-bentuk perintah transfer data 8 bit register ke register selengkapnya dapat dibaca dalam buku *instruction set* mikroprosesor Z-80 CPU.

1.1.2. Transfer data 8 bit dari Memori ke Register

Transfer data 8 bit dari memori ke register adalah transfer data atau copy data 8 bit dari salah satu lokasi memori ke salah satu register 8 bit. Memori bertindak sebagai source dan register bertindak sebagai destinasi. Satu lokasi memori menyimpan data 8bit. Jadi untuk melakukan transfer data 8 bit dari memori ke register cukup satu lokasi memori.

Register 8 bit yang bekerja sebagai destinasi adalah register A, B, C,D,dan E. Untuk menjalankan operasi transfer data 8 bit dari memori ke register diperlukan persyaratan bahwa harus ada mekanisme pemegangan alamat memori. Pemegangan alamat memori biasanya menggunakan register atau alamat langsung. Dalam Z-80 CPU alamat memori ada dua byte atau 16 bit sehingga pemegangan alamat memori menggunakan salah satu register 16 bit yaitu register IX, IY, atau HL. Register IX, IY, atau HL berfungsi sebagai pemegang alamat lokasi memori tempat data source.

Transfer data dari memori dapat terjadi dari lokasi EPROM atau dari lokasi RWM karena kedua memori ini memiliki sifat baca. Untuk operasi transfer data memori keregister ada tanda khusus “( ) “ sebagai penanda operasi memori. Untuk memahami operasi ini lihat Gambar 6.3.

No	Assembly	Operasi	Keterangan
1.	LD A,(1902)	A ← (1902)	isi register A dengan data dari memori lokasi alamat 1902 (RWM)
2.	LD A,(0066)	A ← (0066)	isi register A dengan data dari memori lokasi alamat 0066 (ROM)
3.	LD B, (HL)	B ← (HL)	isi register B dengan data dari memori lokasi alamat sama dengan isi register HL
4.	LD D, (IX+02)	D ←(IX+02)	isi register D dengan data dari memori lokasi alamat sama dengan isi register IX+02

Gambar 6.3. Contoh-contoh transfer data 8 bit memori ke register

Dari Gambar 6.3. pada contoh kasus 1 terjadi proses copy data 8 bit dari memori lokasi atau alamat 1902 ke register A. Memori alamat 1902 berfungsi sebagai source dan register A berfungsi sebagai destinasi. Proses ini memberikan hasil register A berisi data 8 bit dari memori alamat 1902. Untuk kasus nomor 3 regsister B akan dimuati data 8 bit dari memori yang alamatnya dicatat oleh register HL. Misalnya jika regsister HL berisi 1900 maka data dari memori pada alamat 1900 akan di-copy-kan ke register B. Untuk kasus nomor 4 alamat memori sama dengan isi register IX+02. Jika register IX berisi 1900 maka alamat yang diakses adalah 1900+02=1902.

1.1.3. Transfer data Immediate 8 bit ke Register

Data immediate adalah data yang dibangkitkan sendiri bersamaan dengan perintah program.Transfer data immediate 8 bit ke register dapat terjadi terhadap register A, B, C, D, E, H, dan L. Untuk memahami perintah-perintah transfer data immediate 8 bit pelajari contoh Gambar 6.4.

No	Assembly	Operasi	Keterangan
1.	**LD A,19**	A ← 19h	isi register A dengan data 19h
2.	**LD A,00**	A ← 00h	isi register A dengan data 00h
3.	**LD B,3F**	B ← 3Fh	isi register B dengan data 3Fh
4.	**LD C,FF**	C ← FFh	isi register C dengan data FFh

Gambar 6.4. Contoh-contoh transfer data immediate 8 bit ke register

Dari Gambar 6.4. kasus nomor 1 register A sebagai destinasi akan terisi data 19H. Demikian juga untuk kasus nomor 2 register A sebagai destinasi akan termuati data 00H. Pada kasus nomor 3 register B sebagai destinasi akan termuati data 3FH dan pada kasus nomor 4 register C sebagai destinasi akan terisi data FFH. Data *immediate* 19H, 00H,3FH,dan FFH adalah data 8 bit yang dibangkitkan bersamaan dengan perintah program. Transfer data *immediate* sering digunakan dalam program terutama untuk membangkitkan data segera (*immediate*).

1.1.4. Transfer data 8 bit dari Register ke Memori

Transfer data dari register ke memori mencakup persyaratan bahwa harus ada cara atau mekanisme pemegangan alamat memori. Dalam Z-80 CPU alamat memori ada dua byte atau 16 bit seperti telah dijelaskansebelumnya.

Transfer data dari register ke memori dapat terjadi hanya ke lokasi RWM karena ROM tidak bisa diisi data baru. Masalah ini perlu menjadi perhatian khusus karena kesalahan penetapan sasaran memori berarti kegagalan proses perintah transfer data ke lokasi memori. Lokasi memori ROM tidak bisa dijadikan sasaran atau destinasi perintah transfer data register ke memori. Lokasi memori yang bisa dijadikan sasaran adalah RWM. Untuk mengetahui lokasi memori ROM dan RWM diperlukan peta memori. Operasi transfer data dari register ke memori menggunakan tanda “( ) “ sebagai tanda operasi memori. Pemegang alamat lokasi memori menggunakan salah satu register 16 bit atau angka alamat 16 bit.

Transfer data 8 bit dari register ke memori merupakan kebalikan dari transfer data 8 bit dari memori ke register seperti telah diuraikan sebelumnya. Memori sebagai destinasi sering diperlukan untuk menyimpan data-data yang masih diperlukan sementara register yang terbatas jumlahnya harus dikosongkan untuk operasi

berikutnya. Cara kerja transfer data 8 bit dari register ke memori dapat dijelaskan menggunakan Gambar 6.5.

No	Assembly	Operasi	Keterangan
1.	LD(1902), A	(1902) ← A	isi memori lokasi alamat 1902 (RWM) dengan data dari register A
2.	LD (HL), B	(HL) ← B	isi memori lokasi alamat sama dengan isi register HL dengan data dari register B
3.	LD (IX+02), D	(IX+02) ← D	isi memori lokasi alamat sama dengan isi register IX+ 02 dengan data dari register D

Gambar 6.5. Contoh-contoh transfer data 8 bit register ke memori

Dari Gambar 6.5. kasus nomor 1 perintah LD (1902), A akan memberikan proses meng-copy data 8 bit pada register A ke memori alamat 1902. Alamat 1902 adalah alamat RWM dalam mikrokomputer. Sedangkan perintah LD(HL),B menunjukkan proses copy data yang ada di register B ke memori yang alamatnya dicatat oleh register HL. Dalam kasus ini jika HL berisi data 1900 makan data yang ada pada register B akan di-copy-kan ke memori alamat 1900. Untuk kasus nomor 3 sedikit berbeda karena register IX menggunakan indeks. Perintah LD(IX+02),D bekerja meng-copy-kan data yang ada di register D ke memori alamat IX+02.

1.1.5. Transfer data 8 bit dari Memori ke Memori

Transfer data 8 dari memori ke memori terjadi dalam unit memori diluar CPU. Data 8 bit dari satu lokasi memori dicopy kan kelokasi memori lainnya. Transfer data dari memori ke memori memerlukan persyaratan bahwa harus ada mekanisme pemegangan alamat memori. Seperti sudah dibahas sebelumnya dalam Z-80 CPU alamat memori ada dua byte atau 16 bit.

Transfer data dari memori ke memori dapat terjadi diantara semua lokasi RWM dan dari ROM ke RWM. Transfer data ke ROM tidak bisa dilakukan karena ROM tidak bisa diisi data baru. Hal harus menjadi perhatian khusus agar tidak terjadi kegagalan proses. Untuk operasi transfer data dari memori ke memori ada tanda “( ) “ sebagai penanda operasi memori. Untuk menghindari kegagalan perintah maka peta memori sebuah mikrokomputer sangat penting dan diperlukan. Dengan peta memori diperoleh data lokasi dan luasan memori RWM dan ROM secara pasti.

Operasi transfer data dari memori ke memori banyak sekali digunakan untuk memindahkan data atau program dari suatu lokasi memori ke lokasi lain. Disamping juga untuk membuat back-up data dalam suatu lokasi memori memerlukan transfer data dari memori ke memori. Untuk memahami cara kerja transfer data dari memori ke memori pelajari Gambar 6.6.

No	Assembly	Operasi	Keterangan
1.	**LDI**	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1	Transfer 1 byte data dari lokasi memori yang alamatnya dicatat oleh HL ke lokasi memori yang alamatnya dicatat oleh DE
2.	**LDIR**	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	Transfer 1 byte data dari lokasi memori yang alamatnya dicatat oleh HL ke lokasi memori yang alamatnya dicatat oleh DE , Diulang sampai isi reg BC sama dengan nol (alamat naik)
3.	**LDD**	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1	Transfer 1 byte data dari lokasi memori yang alamatnya dicatat oleh HL ke lokasi memori yang alamatnya dicatat oleh DE
4.	**LDDR**	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	Transfer 1 byte data dari lokasi memori yang alamatnya dicatat oleh HL ke lokasi memori yang alamatnya dicatat oleh DE , Diulang sampai isi reg BC sama dengan nol.

Gambar 6.6. Contoh-contoh transfer data 8 bit memori ke memori

Dari Gambar 6.6 kasus nomor LDI adalah mnemonik dari Load Increament. Dalam kolom operasi dapat dibaca terjadi proses transfer data dari memori yang alamatnya dicatat oleh HL ke momori yang alamatnya dicatat oleh DE. Misalnya jika HL berisi 1900H dan DE berisi 1A00H maka akan terjadi transfer data 8 bit dari alamat 1900H ke alamat 1A00H. Bersamaan dengan proses transfer data ini selanjutnya isi register DE bertambah satu menjadi 1A01 dan isi register HL bertambah satu menjadi 1901. Sedangkan register BC berkurang satu. Untuk kasus nomor 2 LDIR adalah singkatan dari *Load Increament Repeat*. Proses yang terjadi sama dengan LDI hanya proses ini berulang sebanyak nilai register BC. Karena register BC adalah register 16 bit maka kemungkinan jumlah pengulangan-nya adalah diantara 1 sampai dengan 65536 kali.

1.1.6. Transfer data Immediate 8 bit ke Memori

Transfer data immediate ke memori menunjukkan transfer data langsung atau segera dari program dengan tujuan/destinasi memori. Kembali hal penting yang harus diperhatikan adalah lokasi memori sebagai destinasi adalah RWM. Untuk lebih mendalami perintah ini pelajari Gambar 6.7. berkut ini.

No	Assembly	Operasi	Keterangan
1.	LD (HL), FF	(HL) ← FF	isi memori lokasi alamat sama dengan isi register HL dengan data FFh
2.	LD (IX+02), 64	(IX+02) ← 64	isi memori lokasi alamat sama dengan isi register IX+ 02 dengan data 64h
3.	LD (IY+02), 19	(IY+02) ← 19	isi memori lokasi alamat sama dengan isi register IY+ 02 dengan data 19h

Gambar 6.7. Contoh-contoh transfer data 8 bit immediate ke memori

Contoh Kasus 1

Berikut ini adalah contoh kasus susunan beberapa perintah transfer data. Identifikasi untuk setiap perintah tergolong kategori transfer data 8 bit yang mana diantara enam jenis perintah transfer data yaitu (1) Register Ke Register, (2) Memori Ke Register, (3) Data Immediate Ke Register, (4) Register Ke Memori, (5) Memori Ke Memori, atau (6) Data Immediate Ke Memori. Dan apa hasil dari masing-masing perintah tersebut. Uraikan proses program Gambar 6.8. di bawah langkah demi langkah sehingga diperoleh hasil seperti Gambar 6.9.

No.	Address	Kode operasi	Assembly	Keterangan	Hasil
1.	1800	3E 93	LD A , 93H	A ← 93H	A=93
2.	1802	47	LD B , A	B ← A	B=93
3.	1803	48	LD C , B	C ← B	C=93
4.	1804	51	LD D , C	D ← C	D=93
5.	1805	5A	LD E , D	E ← D	E=93
6.	1806	26 19	LD H , 19H	H ← 19H	H=19
7.	1808	2E 90	LD L , 90H	L ← 90H	L=90
8.	180A	36 64	LD (HL) , 64H	(HL) ← 64H	(1990)=64
9.	180C	46	LD B , (HL)	B ← (HL)	B=64
10.	180D	3A 90 19	LD A , (1900)	A ← (1900)	A=64
11.	1810	32 91 19	LD (1991) , A	(1991) ← A	(1991)=64
12.	1813	FF	RST 38	STOP	

Gambar 6.8. Contoh program sederhana perintah transfer data 8 bit

Jika dieksekusi penuh kasus program di atas maka maka akan diperoleh hasil seperti Gambar 6.9. berikut.

Reg./Memori	A	F	B	C	D	E	H	L	(1990)	(1991)
Data	64	XX	64	93	93	93	19	90	64	64

Gambar 6.9. Hasil program sederhana perintah transfer data 8 bit

1.2. Transfer Data 16 Bit

Transfer data 16 bit adalah transfer data dengan kapasitas lebih besar dua kali lipat dari transfer data 8 bit. Transfer data 16 bit memiliki lima kemungkinan yaitu:

- Register ke Register,
- Register ke Memori,
- Data Immediate ke Register
- Memori ke Register,
- Memori ke Memori

Untuk membahas lebih lanjut proses transfer data 16 bit berikut disajikan contoh-contoh dan uraian proses transfer data 16 bit.

1.2.1. Transfer data 16 bit dari Register ke Register

Transfer data 16 bit register ke register menunjukkan transfer data dimana source dan destinasi sama-sama register. Transfer data 16 bit register ke register dapat terjadi diantara register 16 bit yaitu register SP, HL, IX, dan IY. Perintah yang digunakan adalah LD (Load). Gambar 6.10. menunjukkan beberapa contoh transfer data 16 bit register ke register. Register 16 bit biasanya digunakan sebagai pemegang alamat memori. Register indeks IX dan IY sangat efektif digunakan untuk memegang alamat memori. Disamping untuk memegang alamat memori register 16 bit HL dapat dioperasikan sebagai akumulator operasi aritmetika 16 bit.

No	Assembly	Operasi	Keterangan
1.	**LD SP,HL**	SP ← HL	isi register SP dengan data dari register HL
2.	**LD SP, IX**	SP ← IX	isi register SP dengan data dari register IX
3.	**LD SP, IY**	SP ← IY	isi register SP dengan data dari register IY

Gambar 6.10. Contoh-contoh transfer data 16 bit register ke register

Dari Gambar 6.10. kasus nomor 1 menjalankan proses meng-copy data pada register HL ke register SP. Register SP adalah register stack pointer dengan kapasitas 16 bit yang bekerja sebagai destinasi.

1.2.2. Transfer data 16 bit dari Memori ke Register

Transfer data dari memori ke register mencakup persyaratan bahwa harus ada cara atau mekanisme pemegangan alamat memori. Dalam Z-80 CPU alamat memori ada dua byte atau 16 bit. Pemegang alamat memori menggunakan salah satu register 16 bit.

Transfer data dari memori dapat terjadi dari lokasi EPROM atau dari lokasi RWM karena kedua memori ini memiliki sifat baca. Untuk operasi ini ada tanda “( ) “ sebagai tanda operasi memori. Transfer data 16 bit dari memori ke register dapat terjadi terhadap register IX, IY, BC, DE, HL, SP,

dan AF. Perintah yang digunakan adalah LOAD dan POP. Gambar 6.11. menunjukkan contoh operasi transfer data 16 bit dari memori ke register.

No	Assembly	Operasi	Keterangan
1.	LD IX,(1902)	$IX_H \leftarrow$ (1903) $IX_L \leftarrow$ (1902)	isi register IX dengan data dari memori lokasi alamat 1903 dan 1902
2.	LD IY,(0066)	$IY_H \leftarrow$ (0067) $IY_L \leftarrow$ (0066)	isi register IY dengan data dari memori lokasi alamat 0067 dan 0066
3.	LD BC,(1800)	B ← (1801) C ← (1800)	isi register BC dengan data dari memori lokasi alamat 1801 dan 1800
4.	POP DE	D ← (SP+1) E ← (SP) SP ← SP+2	isi register DE dengan data dari memori lokasi alamat sama dengan isi register SP+1 dan SP,

Gambar 6.11. Contoh-contoh transfer data 16 bit dari memori ke register

Pada Gambar 6.11. dari contoh kasus 1 perintah LD IX,(1902) memberikan proses transfer data dua kali 8 bit. Data pada alamat 1902 sebagai byte low di-copy-kan ke byte low register IX. Kemudian data pada alamat 1903 sebagai byte high di-copy-kan ke byte high register IX. Dalam kasus ini register IX berisi data 16 bit bersala dari memori alamat 1903 dan alamat 1902. Untuk kasus 2 perintah LD IY,(0066) memberikan proses transfer data dua kali 8 bit. Data pada alamat 0066 sebagai byte low di-copy-kan ke byte low register IY. Kemudian data pada alamat 0067 sebagai byte high di-copy-kan ke byte high register IY. Hasil akhir register IY berisi data 16 bit dari memori alamat 0067 dan 0066. Pada kasus 3 LD BC,(1800) menjalankan proses transfer data dua kali 8 bit. Data pada alamat 1800 sebagai byte low di-copy-kan ke register C. Kemudian data pada alamat 1801 sebagai byte high di-copy-kan ke register B. Hasil akhir register BCberisi data 16 bit dari memori alamat 1801 dan 1800. Pada kasus 4 perintah POP DE memberikan proses transfer data dari memori alamat yang dicatat oelh SP ke register E dan memori dari alamat SP+1 ke register D. Bersamaan perintah ini nilai SP yang baru bertambah dua.

Perintah atau instruksi mikroprosesor banyak yang sulit difahami hanya dengan mencermati perintah assemblynya. Setiap perintah atau instruksi harus difahami lebih detail dengan melihat operasi yang terjadi dengan memperhatikan kolom operasi. Dalam kolom operasi ditunjukkan skema proses yang terjadi untuk setiap instruksi. Setiap instruksi bisa memberikan satu proses, dua proses,tiga proses, bahkan ada yang empat proses. Disamping itu hal lain yang perlu dicermati adalah status flag sebagai akibat dari setiap perintah. Status ini penting sekali untuk beberapa hal dalam pengambilan keputusan dan pemilihan instruksi khusus seperti DAA.

1.2.3. Transfer data Immediate 16 bit ke Register

Transfer data immediate 16 bit ke register adalah pembangkitan data secara langsung ke dalam register melalui data dalam instruksi. Transfer data immediate 16 bit ke register dapat terjadi terhadap register BC, DE, HL, SP, IX, dan IY. Perintah yang digunakan adalah LD (Load). Transfer data immediate 16 bit ke register banyak dan sering digunakan pada pengembangan program. Untuk memahami instruksi transfer data immediate 16 bit ke register perhatikan Gambar 6.12.

No	Assembly	Operasi	Keterangan
1.	**LD BC,1900**	BC ←1900	isi register BC dengan data 1900h
2.	**LD DE,1800**	DE ←1800	isi register DE dengan data 1800h
3.	**LD IX, 203F**	IX ← 203F	isi register IX dengan data 203Fh
4.	**LD IY, EEFF**	IY ← EEFF	isi register IY dengan data EEFFh

Gambar 6.12. Contoh-contoh transfer data immediate 16 bit ke register

Dari Gambar 6.12. pada kasus 1 instruksi LD BC,1900 memberikan proses pengisian register BC dengan data immediate 16 bit bernilai 1900H. Data 16 bit 1900H secara langsung dibangkitkan di register BC. Serupa dengan kasus 1 pada kasus 2 instruksi LD DE,1800 memberikan hasil proses pengisian data 16 bit immediate ke register DE. Perintah ini memberikan hasil register DE=1800H. Pada kasus 3 register IX diisi data immediate 16 bit 203FH. Demikian juga pada kasus 4 menunjukkan instruksi dengan proses pengisian register IY dengan data immediate 16 bit EEFFH. Instruksi-instruksi transfer data immediate 16 bit selengkapnya dapat dilihat dalam *instruction set*. Untuk kasus-kasus yang lain operasinya hampir sama yaitu data 16 yang harus dimasukkan secara langsung.

1.2.4. Transfer data 16 bit dari Register ke Memori

Transfer data 16 bit dari register ke memori adalah transfer data yang melibatkan sebuah register 16 bit dan dua buah lokasi memori karena setiap lokasi memori hanya menyimpan 8 bit data. Transfer data dari register ke memori memerlukan persyaratan bahwa harus ada cara atau mekanisme pemegangan alamat memori. Dalam Z-80 CPU alamat memori ada dua byte atau 16 bit.

Transfer data dari register ke memori dapat terjadi hanya ke lokasi RWM karena ROM tidak bisa diisi data baru. Untuk operasi ini ada tanda “( )“ sebagai tanda

operasi memori menggunakan salah satu register 16 bit atau angka alamat.

Transfer data 16 bit dari register ke memori dapat terjadi drai register BC, DE, HL, IX, IY, dan AF. Perintah yang digunakan adalah LD (Load) dan PUSH. Untuk memahami perintah transfer data 16 bit dari register ke memori perhatikan Gambar 6.13.

Dari Gambar 6.13. untuk kasus 1 instruksi LD(1902),BC menunjukkan instruksi berhubungan dengan memori karena ada (1902). Dalam instruksi ini ada dua proses yang terjadi yaitu proses pertama meng-copy data pada register C ke memori alamat 1902 dan kedua proses meng-copy data pada register B ke memori alamat 1903. Kedua proses transfer data ini menghasilkan transfer data 16 bit di dua lokasi memori yaitu alamat 1902 dan 1903. Untuk kasus ke dua instruksi LD (1800),HL melakukan proses isi register H di-copy ke memori alamat 1801 dan register L di-copy ke memori alamat 1800. Hasil akhir adalah transfer data 16 bit dari register HL ke memori alamat 1800 dan 1801. Pada kasus ke 3 PUSH IX melakukan tiga proses yaitu memasukan data dari register IX low ke memori alamat SP-1dan memasukkan data dari register IX high ke memori alamat SP-2. Dalam proses ini selanjutnya isi register SP yang baru SP semula kurang dua. Perintah PUSH adalah instruksi yang biasa dipasangkan dengan instruksi POP. Kedua perintah ini digunakan pada operasi pembentukan stack pointer. Sehingga selalu berhubungan dengan register SP (Stack Pointer). PUSH mendorong data ke memori sehingga termasuk kategori transfer data 16 bit dari register ke memori. Sedangkan instruksi POP bekerja menarik data 16 bit dari dua lokasi memori ke register 16 bit.

No	Assembly	Operasi	Keterangan
1.	LD(1902), BC	**(1903) ← B** **(1902) ← C**	Isi memori alamat 1902 (RWM) dengan data dari register C dan memori alamat 1903 dengan data register B
2.	LD (1800), HL	**(1801) ← H** **(1800) ← L**	Isi memori alamat 1800 dengan isi register L dan alamat 1801 dengan data dari register H
3.	PUSH IX	**(SP-1) ← IX_H** **(SP-2) ← IX_L** **SP ← SP-2**	Isi memori alamat sama dengan isi register SP-1 dengan data dari register IX_H dan SP-2 dengan data dari register IX_L

Gambar 6.13. Contoh-contoh transfer data 16 bit dari register ke memori

1.2.5 Transfer data 16 bit dari Memori ke Memori

Transfer data 16 bit dari memori ke memori merupakan kelompok instruksi yang bekerja meng-copy data dari dua lokasi memori source ke dua lokasi memori destinasi. Transfer data dari memori ke memori juga membutuhkan persyaratan bahwa harus ada mekanisme pemegangan alamat memori. Dalam Z-80 CPU alamat memori ada dua byte atau 16 bit.

Transfer data dari memori ke memori dapat terjadi hanya ke lokasi RWM atau dari ROM ke RWM karena ROM tidak bisa diisi data baru. Untuk operasi ini ada tanda “( ) “ sebagai tanda operasi memori. Perintah yang dapat digunakan hanya LDIR (load increament repeat) dan LDDR (load decreament repeat). Dalam hal ini jumlah byte data yang dapat ditransfer dua byte atau lebih dengan kemampuan maksimum 64 K byte bergantung isi register BC.

Jenis instruksi LDIR dan LDDR sangat baik digunakan untuk memindahkan atau meng-copy sejumlah data dari suatu lokasi memori kelokasi memori lain untuk keperluan penggandaan data dari suatu blok memori ke blok lokasi memori lainnya. Untuk memahami transfer data 16 bit dari memori ke memori dapat digambarkan seperti Gambar 6.14.

No	Assembly	Operasi	Keterangan
1.	**LDIR**	(DE) ← (HL) DE ← DE+1 HL ← HL+1 BC ← BC-1 Repeat until BC = 0	Transfer 1 byte data dari lokasi memori yang alamatnya dicatat oleh HL ke lokasi memori yang alamatnya dicatat oleh DE , Diulang sampai isi reg BC sama dengan nol (alamat naik)
2.	**LDDR**	(DE) ← (HL) DE ← DE-1 HL ← HL-1 BC ← BC-1 Repeat until BC = 0	Transfer 1 byte data dari lokasi memori yang alamatnya dicatat oleh HL ke lokasi memori yang alamatnya dicatat oleh DE , Diulang sampai isi reg BC sama dengan nol (alamat turun)

Gambar 6.14. Contoh-contoh transfer data 16 bit dari memori ke memori

Instruksi LDIR melakukan operasi copy data dari satu lokasi memori yang alamatnya dicatat oleh register HL ke satu lokasi memori yang alamatnya dicatat oleh regsister DE. Selanjutnya terjadi proses penambahan nilai register DE=DE+1, HL=HL+1, dan BC=BC-1. Proses ini akan berulang hingga isi register BC=0000H. Dengan demikian sangat jelas jumlah byte data yang di-copy bisa diatur dengan seting nilai data pada register BC. Jika diinginkan ada 10 byte data yang di-copy maka register BC diisi data 000AH. LDIR bekerja dari alamat kecil ke alamat besar. Instruksi LDDR bekerja hampir sama dengan LDIR. Perbedaannya terletak pada proses LDIR dari memori besar ke kecil.

Contoh Kasus 2

Gambar 6.15 berikut ini adalah susunan beberapa perintah transfer data 16 bit. Identifikasi untuk setiap perintah tergolong kategori transfer data 16 bit yang mana dan apa hasil dari masing-masing perintah tersebut.

No.	Address	Kode operasi	Mnemonic	Keterangan
1.	1820	01 63 19	LD BC, 1963H	BC ← 1963H
2.	1823	11 64 19	LD DE, 1964H	DE ← 1964H
3.	1826	21 95 19	LD HL, 1995H	HL ← 1995H
4.	1829	31 91 19	LD SP, 1991H	SP ← 1991H
5.	182C	ED 43 93 19	LD (1993), BC	(1993) ← C
6.				(1994) ← B
7.	1830	ED 53 95 19	LD (1995), DE	(1995) ← E
8.				(1996) ← D
9.	1834	DD 21 00 18	LD IX, 1800H	IX ← 1800H
10.	1838	FD 21 13 18	LD IY, 1813H	IY ← 1813H
11.	183C	DD E5	PUSH IX	(SP-2) ← IXL
12.				(SP-1) ← IXH
13.				SP ← SP-2
14.	183E	FD E5	PUSH IY	(SP-2) ← IYL
15.				(SP-1) ← IYH
16.				SP ← SP-2
17.	1840	DD E1	POP IX	IXH ← (SP+1)
18.				IXL ← (SP)
19.				SP ← SP+2
20.	1842	FD E1	POP IY	IYH ← (SP+1)
21.				IYL ← (SP)
22.				SP ← SP+2
23.	1844	FF	RST 38	STOP

Gambar 6.15a. Kasus program sederhana perintah transfer data 16 bit

Dengan menguraikan proses program Gambar 6.15a. langkah demi langkah akan diperoleh hasil-hasil akhir seperti Gambar 6.15b.

Reg./Memori	BC	DE	HL	SP	IX	IY	198D	198E	198F	1990
Data	1963	1964	1995	1991	1800	1813	13	18	00	18

Gambar 6.15b. Hasil kasus program sederhana perintah transfer data 16 bit

2. Pertukaran Data

Pertukaran data berbeda dengan transfer data. Pertukaran data bekerja saling meng-copy diantara dua lokasi register atau memori. Pertukaran data dapat dilakukan diantara dua register, kelompok pasangan register dan antara register dengan memori. Jadi sangat jelas pertukaran data berbeda dengan transfer data.

Instruksi yang digunakan untuk pertukaran data adalah EX (exchange) dan EXX (exchange). Perintah EX dan EXX memiliki perbedaan dimana EX hanya untuk satu register 16 bit. Sedangkan perintah EXX bekerja untuk tiga kelompok register 16 bit yaitu diantara register utama dan register alternatif.

Pertukaran data dapat terjadi diantara register DE, HL, BC, BC', DE', HL', memori, memori dengan register. Pada Gambar 6.17. ditunjukkan sejumlah contoh instruksi pertukaran data. Dari Gambar 6.16. instruksi EX DE,HL bekerja mempertukarkan data 16 bit di register HL dengan data 16 bit di register DE. Hasilnya register HL yang baru berisi data 16 bit dari register DE dan register DE akan berisi data baru dari register HL.

Instruksi EX AF,AF' bekerja mempertukarkan isi register utama AF dengan isi register alternatif AF'. Jika ingin mempertukarkan seluruh data yang ada pada register utama BC, DE, HL dengan register alternatif BC', DE', HL' maka instruksinya adalah EXX.

Instruksi EX (SP),HL bekerja melakukan pertukaran data 16 bit diantara memori yang alamatnya dicatat oleh SP dan SP+1 dengan register HL. Data yang ada pada lokasi memori alamat (SP) dipertukarkan dengan data 8 bit di register L dan data 8 bit di memori alamat (SP+1) dipertukarkan dengan data 8 bit di register H.

Instruksi EX (SP),IX melakukan proses pertukaran data yang sama dengan proses EX (SP),HL. Bedanya terletak pada jenis registernya yaitu register IX .

Assembly	Operasi	Jenis Transfer Data
EX DE, HL	DE ←→ HL	Reg Reg
EX AF , AF'	AF ←→ AF'	Reg
EXX	BC ←→ BC'	Reg
	DE ←→ DE'	
	HL ←→ HL'	
EX (SP) , HL	H←→ (SP+1)	
	L ←→ (SP)	Reg Memori
EX (SP) , IX	IXH←→(SP+1)	
	IXL ←→ (SP)	

Gambar 6.16. Contoh-contoh instruksi pertukaran data

Contoh Kasus 3

Untuk melatih dan meningkatkan pemahaman maka analisislah program Gambar 6.17a berikut ini dan tentukan hasil untuk setiap perintah !!

No.	Address	Kode operasi	Assembly	Keterangan	HASIL
1.	1800	11 01 01	LD DE , 0101H	DE ← 0101H	DE = 0101
2.	1803	21 FF FF	LD HL , FFFFH	HL ← FFFFH	HL = FFFF
3.	1806	EB	EX DE , HL	HL ←→ DE	DE=FFFF ; HL=0101
4.	1807	01 02 02	LD BC , 0202H	BC ← 0202H	BC = 0202
5.	180A	11 03 03	LD DE , 0303H	DE ← 0303H	DE = 0303.
6.	180D	78	LD A , B	A ← B	A = 02
7.	180E	08	EX AF , AF'	AF ←→ AF'	AF=XXXX; AF'=02XX
8.	180F	31 90 19	LD SP , 1990H	SP ← 1990H	SP = 1990
9.	1812	E3	EX (SP) , HL	H ←→ (SP+1)	HL = XXXX
10.				L ←→ (SP)	(1990) = FF; (1991)=FF
11.	1813	DD 21 22 00	LD IX , 0022H	IX ← 0022H	IX = 0022
12.	1817	31 92 19	LD SP , 1992H	SP ← 1992H	SP = 1992
13.	181A	DD E3	EX (SP) , IX	IXH ←→ (SP+1)	IX = FFFF
14.				IXL ←→ (SP)	(1990) = 00; (1991)=22
15.	181C	FF	RST 38	STOP	
16.					

Gambar 6.17a. Contoh kasus program sederhana pertukaran data

Dengan menguraikan proses program Gambar 6.17a. langkah demi langkah akan diperoleh hasil-hasil akhir seperti Gambar 6.17b.

Reg./Memori	BC	DE	HL	SP	IX	IY	198D	198E	1990	1991
Data	0202	0303	XXXX	1992	FFFF	XXXX	13	18	00	22

Gambar 6.17b. Hasil kasus program sederhana perintah transfer data 16 bit

3. Pelacakan Data

Pelacakan atau searching data sangat diperlukan dalam pengembangan program. Pelacakan atau searching data dilakukan dengan perintah *compare* atau bandingkan. Dalam mikroprosesor instruksi membading kan dilakukan dengan mengurangkan nilai bilangan yang dibandingkan dengan bilangan pembanding. Instruksi perbandingan tidak merubah nilai yang dibanding dengan nilai bilangan pembanding. Dalam Z-80 CPU pembandingan merujuk ke register A.

Untuk mengetahui nilai sebuah data dalam suatu lokasi memori perlu melakukan pembandingan dan pelacakan. Misalnya dalam 100 lokasi memori jika diinginkan untuk mengetahui jumlah data bernilai 90 dari 100 data dapat dilakukan dengan instruksi ini. Untuk keperluan lain jika ingin mengetahui lokasi sebuah data juga dapat menggunakan jenis instruksi ini. Untuk mengetahui nilai sebuah data pada suatu lokasi memori atau menemukan ada tidaknya sebuah data bernilai tertentu dari sekelompok data dapat ditempuh dengan melakukan searching. Perintah search yang digunakan adalah CPI (compare increament), CPIR (compare increament repeat), CPD (compare decreament), dan CPDR (compare decreament repeat). Untuk memahami instruksi pelacakan data pelajari Gambar 6.18.

Assembly	Operasi	Keterangan
CPI	A - (HL) HL ←HL + 1 BC ←BC – 1	Bandingkan isi A dengan data di memori lokasi alamat dicatat HL. Alamat memori oleh HL naik
CPIR	A - (HL) HL ← HL + 1 BC ←BC – 1 Repeat until A = (HL) or BC=0	Bandingkan isi A dengan data di memori lokasi alamat dicatat HL. Berhenti sampai nilai A=(HL) atau BC = 0. Alamat memori oleh HL naik
CPD	A - (HL) HL ← HL - 1 BC ← BC – 1	Bandingkan isi A dengan data di memori lokasi alamat dicatat HL. Alamat memori oleh HL turun
CPDR	A - (HL) HL ← HL - 1 BC ← BC – 1 Repeat until A = (HL) or BC=0	Bandingkan isi A dengan data di memori lokasi alamat dicatat HL. Berhenti sampai nilai A=(HL) atau BC = 0. Alamat memori oleh HL turun

Gambar 6.18. Contoh-contoh instuksi pelacakan data

Instruksi CPI bekerja membandingkan data register A dengan data yang ada di memori yang alamatnya dicatat oleh register HL. Perbandingan dua buah data akan memberikan status carry = 1 jika data di register A lebih kecil dari data memori alamat dicatat oleh register HL. Untuk keadaan lai carry = 0 jika data di register A sama atau lebih besar dari data memori alamat dicatat oleh register HL dan zerro flag Z=1 jika data di register A sama dengan data memori alamat dicatat oleh register HL.Selanjutnya terjadi juga penambahan HL=HL+1 dan pengurangan BC= BC-1.

Instruksi CPIR menunjukkan tipe instruksi yang lebih jelas dalam melakukan proses pelacakan data. Dalam operasi sangat jelas terjadi proses membandingkan data yang ada pada register A dengan data yang ada pada suatu lokasi memori yang alamatnya dicatat oleh register HL. Data register HL kemudian ditambah 1 dan register BC datanya berangsur dikurangi 1. Proses pembandingan dihentikan jika nilai data di register A sama dengan data di memori yang alamatnya dicatat oleh register HL. Instruksi CPIR bekerja melacak data 8 bit di memori melalui register A. Lokasi alamat data yang dilacak tercatat pada register HL. Pelacakan data berlangsung terus sampai ditemukan data yang sama dengan data yang ada di register A atau berhenti jika isi register BC=0000. Jadi register BC dapat digunakan untuk menetapkan luasan atau jumlah data yang dilacak di dalam memori.

Instruksu CPIR bekerja maju menuju alamat diatasnya. Karena isi register HL bertambah satu. Pelacakan dimulai dari alamat rendah kealamat yang lebih tinggi sampai dinyatakan diketemukan atau tidak diketemukan tetapi cacahan register BC=0000. Untuk kasus instruksi CPD dan CPDR mekanisme kerja proses pelacakan data sama dengan CPIR. Bedanya pada CPDR pelacakan dimulai dari alamat tinggi ke alamat rendah.

4. Instruksi Aritmetika

Dalam mikroprosesor Zilog Z-80 CPU instruksi-instruksi aritmetika yang disediakan jumlahnya terbatas pada instruksi penjumlahan (**ADD** dan **ADC**) dan pengurangan (**SUB** dan **SBC**) saja. Dengan demikian untuk memecahkan persoalan aritmetika lainnya seperti perkalian dan pembagian tidak dapat diselesaikan secara langsung dengan satu buah instruksi. Dengan menggabungkan beberapa instruksi yang tersedia dapat dibuat program subroutin untuk perkalian dan pembagian, mencari nilai kuadrat suatu bilangan, sortir data, pengurutan, dan sebagainya.

Perlu diingat bahwa mikroprosesor melalukan operasi penjumlahan dan pengurangan dalam sistim bilangan biner. Mikroprosesor menyediakan layanan operasi baik untuk bilangan tidak bertanda maupun bilangan bertanda. Pada saat bekerja dalam bilangan bertanda user harus mampu memaknai nilai sebuah data. Dalam operasi bilangan bertanda digunakan komplemen dua. Untuk keperluan tertentu juga dibutuhkan bekerja dalam sistim bilangan desimal hampir di segala bidang. Untuk itu dalam operasi aritmetika disediakan instruksi Decimal Addjust Accumulator (DAA) untuk memberikan faktor koreksi pada saat kita bekerja dalam sistim bilangan desimal dalam kode BCD.

Instruksi CP,s disediakan untuk membandingkan isi akumulator dengan sebuah data tanpa merubah isi akumulator. Instruksi ini memberikan akibat pada perubahan register flag sebagai status pembandingannya. Status tersebut diantaranya adalah (S=Sign, Z=Zerro, H=Half Carry, dan C=Carry). Dalam melaksanakan instruksi pembandingan, mikroprosesor menggunakkan sistim bilangan komplemen dua.

Pada sistim komplemen dua bilangan terkecil adalah 80H = 1000 0000B = -128 dan bilangan terbesar adalah 7F = 0111 1111 = +127. Bit 7 digunakan sebagai tanda bilangan. Jika bit 7=0 menunjukkan nilai positif dan jika bit 7= 1 menunjukkan bilangan bernilai getatif.

4.1. Instruksi ADD dan SUB.

Instruksi ADD digunakan untuk melakukan operasi penjumlahan 8 bit dan 16 bit. Ada 38 jenis perintah penjumlahan pada mikroprosesor Z-80 CPU. Pada operasi 8 bit register A (akumulator) ditambahkan dengan isi sebuah register 8 bit atau data immediate 8 bit, atau data pada satu lokasi memori yang alamatnya dicatat oleh register HL, IX, atau IY.

Pada operasi aritmetika 16 bit register HL, iX, dan IY berfungsi sebagai akumulator yang dapat ditambahkan dengan isi register BC, DE, HL, SP. Untuk lebih jelasnya perhatikan Gambar 6.19. berikut.

Operasi	Assembly	Operasi	Ket.
8 Bit	ADD A , A	A ← A + A	• Mempe-ngaruhi Flag S, Z, H, V, C • N= data 8 bit
	ADD A , B	A ← A + B	
	ADD A , C	A ← A + C	
	ADD A , D	A ← A + D	
	ADD A, E	A ← A + E	
	ADD A , H	A ← A + H	
	ADD A , L	A ← A + L	
	ADD A, N	A ← A + N	
	ADD A, (HL)	A← A + (HL)	
	ADD A,(IX+d)	A ← A + IX+d)	
	ADD A , (IY+d)	A ← A+(IY+d)	
16 Blt	ADD HL , BC	HL ← HL + BC	
	ADD HL , DE	HL ←HL + DE	
	ADD HL , HL	HL ←HL + HL	Hanya Mempenga ruhi Flag carry
	ADD HL , SP	HL ←HL + SP	
	ADD IX, BC	IX ← IX + BC	
	ADD IX , DE	IX ← IX + DE	
	ADD IX , IX	IX ← IX + IX	
	ADD IX, SP	IX ← IX + SP	
	ADD IY , BC	IY ← IY + BC	
	ADD IY , DE	IY ← IY + DE	
	ADD IY , IY	IY ← IY + IY	
	ADD IY , SP	IY ← IY + SP	

Gambar 6.19. Contoh-contoh instruksi ADD

Pada Gambar 6.18. tercatat ada dua kelompok instruksi ADD yaitu kelompok 8 bit dan kelompok16 bit. Dalam kelompok 8 bit semua hasil operasi ADD tercatat di register A. Register A sebagai penampung hasil atau lebih umum disebut dengan akumulator. Dalam kelompok 16 bit register HL, IX,dan

IY salah satu bisa berfungsi sebagai akumulator.

Instruksi ADD A,A akan menjalankan proses menjumlahkan data yang ada di register A dengan dirinya sendiri. Sehiingga instruksi ini mewakili perkalian register A dengan 2. Instruksi ADD A, B melakukan perintah penjumlahan data yang ada di A dengan data yang ada di B dan hasilnya dicatat di register A. Demikian juga dengan instruksi ADD yang lain yang terjadi antara register A dengan register 8 bit lainnya. Instruksi ADD A,N menunjukkan instruksi penjumlahan data yang ada di register A dengan data immediate dan hasilnya dicatat di register A. Instruksi ADD A, (HL) menjalankan perintah penjumlahan data yang ada di A dengan data yang ada di memori yang alamatnya dicatat oleh register HL. Dua perintah dibawahnya identik terhadap memori yang alamatnya dicatat oleh register IX dan IY.

Penjumlahan 16 bit dengan instruksi ADD HL, BC bekerja menjalankan perintah penjumlahan data 16 bit di register HL dengan data 16 bit di register BC dan hasilnya dicatat di register HL. Instruksi ADD HL, DE bekerja menjalankan perintah penjumlahan data 16 bit di register HL dengan data 16 bit di register DE dan hasilnya dicatat di register HL. Untuk instruksi yang lain penjumlahannya bekerja identik.

Instruksi SUB digunakan hanya untuk melakukan operasi pengurangan 8 bit. Pada operasi SUB isi register A dikurangkan dengan salah satu isi register A, B, C, D, E, H, L, atau data immediate 8 bit. Disamping itu isi register A juga dapat dikurangkan dengan data pada suatu lokasi memori yang alamatnya dicatat oleh register HL, IX, dan IY. Gambar 6.20. menunjukkan contoh-contoh perintah SUB.

Operasi	Assembly	Operasi	Keterangan
8 Bit	SUB , A SUB , B SUB , C SUB , D SUB, E SUB , H SUB , L SUB, N SUB , HL) SUB, (IX+d) SUB,(IY+d)	A ← A- A A ← A- B A ← A- C A ← A- D A ←A- E A ←A- H A ← A - L A ← A- N A←A-(HL) A ←A-(IX+d) A ←A-(IY+d)	Mempengaruhi Flag S, Z, H, V, C N = data 8 bit

Gambar 6.20. Contoh-contoh perintah SUB

Semua instruksi ADD dan SUB dapat mempengaruhi status flag yaitu Sign, Zerro, Half Carry, Overflow, dan Carry pada Register F. Pada instruksi ADD akan terjadi flag N=0 dan pada instruksi SUB akan terjadi flag N=1. Dua keadaan ini digunakan untuk menyatakan fungsi flag C sebagai carry atau borrow**.** Flag C bermakna Carry jika operasi aritmetika itu adalah ADD dan flag C bermakna borrow jika operasi aritmetika itu adalah SUB.

4.2. Instruksi ADC (ADD With Carry) dan SBC (Sub With Carry)

Instruksi ADC digunakan untuk menambahkan isi register A dengan data 8 bit yang berada pada suatu register atau data immediate atau data suatu lokasi memori dan mengikut sertakan bit Carry (C) yang ada di register F. Instruksi ADC juga digunakan untuk menambahkan isi register HL dengan data 16 bit yang berada pada register BC, DE, HL, dan SP dengan mengikut sertakan bit Carry Flag (C). Beberapa contoh instruksi ADC dapat dipelajari dari Gambar 6.20. Pada Gambar 6.20 dapat dilihat instruksi ADC juga menggunakan register A sebagai penampung atau akumulator untuk operasi 8 bit dan register HL untuk operasi 16 bit.

Operasi	Assembly	Operasi	Keterangan
8 Bit	ADC A , A	A ←A + A + Cy	Mempengaruhi Flag S, Z, H, V, C N = data 8 bit
	ADC A , B	A ← A + B + Cy	
	ADC A , C	A ← A + C + Cy	
	ADC A , D	A ← A + D + Cy	
	ADC A, E	A ← A + E + Cy	
	ADC A , H	A ← A + H + Cy	
	ADC A , L	A ← A + L + Cy	
	ADC A, N	A ← A + N + Cy	
	ADC A , (HL)	A ← A + (HL)+Cy	
	ADC A, (IX+d)	A ←A + (IX+d)+Cy	
	ADC A, (IY+d)	A ←A + (IY+d)+Cy	
16 BIt	ADC HL , BC	HL← HL+ BC + Cy	Hanya Mempengaruhi Flag carry
	ADC HL , DE	HL← HL+ DE + Cy	
	ADC HL , HL	HL← HL + HL+ Cy	
	ADC HL , SP	HL←HL + SP + Cy	

Gambar 6.21. Contoh-contoh instruksi ADC

Instruksi SBC digunakan untuk mengurangkan isi register A dengan data 8 bit yang berada pada suatu register atau data immediate atau data suatu lokasi memori dengan mengikutsertakan bit carry flag. Instruksi SBC juga digunakan untuk mengurangkan isi register HL dengan data 16 bit yang berada pada register BC, DE, HL, dan SP dengan mengikutsertakan bit Carry Flag (Cy). Hasil dari kedua bentuk pengurangan tersebut dicatat di Register A atau Register HL. Gambar 6.22. menunjukkan beberapa contoh instruksi SBC. Untuk operasi 8 bit register A sebagai penampung hasil dan untuk operasi 16 bit register HL sebagai penampung hasil. Karena register A sebagai penampung hasil maka disebut juga akumulator.

Operasi	Assembly	Operasi	Keterangan
8 Bit	SBC A , A	A ←A - A - Cy	Mempengaruhi Flag S, Z, H, V, C N = data 8 bit
	SBC A , B	A ←A - B - Cy	
	SBC A , C	A ←A - C - Cy	
	SBC A , D	A ← A- D - Cy	
	SBC A, E	A ←A - E - Cy	
	SBC A , H	A ←A - H - Cy	
	SBC A , L	A ← A- L - Cy	
	SBC A, N	A ← A- N - Cy	
	SBC A, (HL)	A←A- (HL)-Cy	
	SBC A, (IX+d)	A←A- (IX-d)-Cy	
	SBC A, (IY+d)	A ←A- (IY-d)-Cy	
16 BIt	SBC HL, BC	HL← HL- BC-Cy	Hanya Mempengaruhi Flag carry
	SBC HL, DE	HL← HL- DE- Cy	
	SBC HL HL	HL ←HL- HL- Cy	
	SBC HL, SP	HL ←HL-SP- Cy	

Gambar 6.22. Contoh-contoh instruksi SBC

4.3. Instruksi INC (Increment) dan DEC (Decrement)

Instruksi INC digunakan untuk menambah isi suatu register atau memori dengan satu nilai. Instruksi ini sangat potensial digunakan untuk membuat counter cacah naik.

Instruksi INC dapat terjadi pada data yang tersimpan di register 8 bit, register 16 bit, dan data memori yang alamatnya dicatat oleh HL, IX, dan IY. Pada Gambar 6.23. ditunjukkan contoh-contoh instruksi INC lengkap dengan simboloperasinya.

Operasi	**Assembly**	**Operasi**	Keterang - an
8 Bit	INC A	A ← A + 1	Mempe-ngaruhi Flag S, Z, H, V, C
	INC B	B ←B + 1	
	INC C	C ←C + 1	
	INC D	D ← D + 1	
	INC E	E ← E + 1	
	INC H	H ←H + 1	
	INC L	L ←L + 1	
16 Bit	INC BC	BC ← BC + 1	
	INC DE	DE ←DE + 1	
	INC HL	HL ←HL + 1	
	INC IX	IX ← IX + 1	
	INC IY	IY ← IY + 1	
	INC SP	SP ←SP + 1	
Memori	INC (HL)	(HL) ←(HL)+1	
	INC (IX+d)	(IX+d) ←(IX+d) +1	
	INC (IY+d)	(IY+d) ←(IY+d) +1	

Gambar 6.23. Contoh-contoh instruksi INC

Instruksi INC dapat terjadi terhadap register 8 bit, register 16 bit, dan data pada memori.

Instruksi DEC digunakan untuk mengurangi data suatu register atau data suatu memori dengan 1. Pada Gambar 6.24.ditunjukkan beberapa contoh instruksi DEC

Operasi	Assembly	Operasi	Ketera-ngan
8 Bit	DEC A	A ← A - 1	Mempe-ngaruhi Flag S, Z, H, V, C
	DEC B	B ← B - 1	
	DEC C	C ← C - 1	
	DEC D	D ← D - 1	
	DEC E	E ← E - 1	
	DEC H	H ← H - 1	
	DEC L	L ← L - 1	
16 Bit	DEC BC	BC ← BC - 1	
	DEC DE	DE ← DE - 1	
	DEC HL	HL ← HL - 1	
	DEC IX	IX ← IX - 1	
	DEC IY	IY ← IY - 1	
	DEC SP	SP ← SP - 1	
Memori	DEC (HL)	(HL) ←(HL)- 1	
	DEC (IX+d)	(IX+d)←(IX+d)-1	
	DEC (IY+d)	(IY+d)←(IY+d)-1	

Gambar 6.24. Contoh-contoh instruksi DEC

Insruksi DEC dapat terjadi terhadap data yang ada pada register 8 bit, register 16 bit dan data pada suatu lokasi memori. Instruksi DEC banyak sekali digunakan untuk keperluan pengaturan proses iterasi. Dengan mengeset isi sebuah register lalu mengurangi dengan satu secara berulang-ulang proses iterasi suatu proses berulang dapat dijalankan dengan efektif. Pada contoh-contoh kasus program nantinya akan banyak dapat disaksikan.

- **Instruksi Aritmetika Khusus**

Dalam operasi aritmetika disediakan beberapa instruksi khusus yaitu :

- DAA mnemonic dari Decimal Adjust Accumulator
- CPL mnemonic dari Complement Accumulator (Komplemen 1)
- NEG mnemonic dari Negate Accumulator (Komplemen 2)

4.4. Instruksi DAA

Instruksi DAA digunakan untuk merubah isi register A ke bentuk BCD. Instruksi DAA digunakan untuk memberi faktor koreksi pada saat bekerja dengan bilangan desimal.

DAA dalam melakukan koreksi bekerja sbb :

Jika Bit b3, b2, b1, b0 > 9 atau ada Half Carry (H = 1) maka bit b3, b2, b1, b0 ditambah dengan 0110 = 6.
Jika Bit b7, b6, b5, b4 > 9 atau ada Carry (C = 1) maka bit b7, b6, b5, b4 ditanbah dengan 0110 = 6.

Contoh .

Desimal	Biner
29_{10}	0010 1001
26_{10}	0010 0110
---------- +	
55_{10}	0100 1111= 4FH

Nilai hasil BCD yang seharusnya 55 ternyata tidak sesuai dengan hasil penjumlahan BCD yaitu 0100 1111 = 4F. Ini memerlukan koreksi dengan perintah DAA. Karena bit B3,B2,B1,B0 lebih besar dari 9 maka Koreksi DAA ditambahkan dengan 06.

Koreksi DAA

```
  0100  1111= 4F
  0000  0110= 06
------------------ +
  0101  0101  = 55
```

Koreksi DAA telah memberikan nilai hasil penjumlahan yang benar yaitu 55. Dalam sistem mikroprosesor bentuk koreksi DAA dijalankan secara langsung pada saat program di eksekusi. Untuk itu pada setiap operasi aritmetika bilangan desimal harus selalu disertakan koreksi DAA untuk memberi faktor koreksi jika diperlukan.

4.5. Instruksi CPL (Complement)

Instruksi CPL digunakan untuk merubah isi akumulator menjadi bentuk komplemen 1 yaitu dengan menginverse semua bit yang ada di akumulator. CPL sama artinya dengan NOT.

A ← NOT A

3.6. Instruksi NEG (Negate)

Instruksi NEG digunakan untuk merubah isi akumulator menjadi bentuk negatifnya yaitu dengan merubahnya menjadi nilai komplemen dua.

A ← NOT A + 1

Ini sama dengan komplemen dua yaitu komplemen satu ditambah 1.

4.7. Instruksi CP (Compare)

Digunakan untuk membandingkan isi akumulator dengan data immediate 8 bit atau isi salah satu register 8 bit atau isi/data suatu lokasi memori ***tanpa merubah isi akumulator.*** Instruksi CP membangun keadaan pada status Flag pada Bit Sign, Zero, Over Flow, Half Carry dan Carry pada Register Flag. Instruksi CP sangat baik digunakan untuk menguji sebuah data apakah data tersebut sama dengan suatu nilai tertentu atau lebih atau lebih kecil dari suatu nilai tertentu. Beberapa contoh instruksi CP dijabarkan pada Gambar 6.25.

Operasi	Assembly	Operasi	Keterangan
8 Bit	CP A CP B CP C CP D CP E CP H CP L CP N CP (HL) CP (IY + d) CP (IX + d)	A - A A - B A - C A - D A - E A - H A - L A - N A - (HL) A - (IY + d) A - (IX + d)	Mempengaruhi Flag S, Z, H, V, C Nilai A tetap atau tidak berubah

Gambar 6.25. Contoh-contoh instruksi CP

Instruksi CP bekerja membandingkan isi register A dengan register lain dengan cara mengurangkan tanpa merubah data register A. Misalnya CP B akan bekerja mengurangkan data di register A dengan data di register B tanpa merubah data di register A.

5. Instruksi LOGIKA AND, OR, dan XOR

Instruksi AND, OR, dan XOR digunakan untuk melakukan operasi logika isi dari akumulator terhadap data suatu register 8 bit atau data immediate, atau data suatu lokasi memori.

Operasi	Assembly	Operasi	Keterangan
8 Bit	AND A AND B AND C AND D AND E AND H AND L	A ← A ^ A A ← A ^ B A ← A ^ C A ← A ^ D A ← A ^ E A ← A ^ H A ← A ^ L	Mempengaruhi Flag S, Z, H, V, C
memori	AND (HL) AND (IX+d) AND (IY+d)	A ← A ^(HL) A ← A^(IX +d) A← A ^(IY +d)	

Gambar 6.26. Contoh-contoh instruksi logika

Pola di atas berlaku juga pada operasi LOGIKA OR dan XOR. Simbol operasi Logika adalah sbb :

^ : untuk LOGIKA AND

V : untuk LOGIKA OR

⊕ : untuk LOGIKA XOR

Untuk meningkatkan pemahaman dan kemampuan memprogram mikroprosesor pada bagian berikut akan dijabarkan beberapa kasus dan pemecahan masalah untuk setiap kasus.

Contoh Kasus 4

Kasus 4 menjabarkan Algoritma Program, Flow Chart dan Program permasalahan menjumlahkan 2 byte bilangan biner yang masing-masing beralamat di 1900H, 1901H, 1902H, 1903H. Hasilnya di simpan pada alamat 1904H, 1905H, 1906H. Program di tulis pada alamat awal program 1800H.

Masalah penjumlahan 2 byte bilangan biner dapat digambarkan seperi Gambar 6.27. berikut:

```
          [ 1901 ]   [ 1900 ]
          [ 1903 ]   [ 1902 ]
        ------------------------- +
[ 1906 ]   [ 1905 ]   [ 1904 ]
```

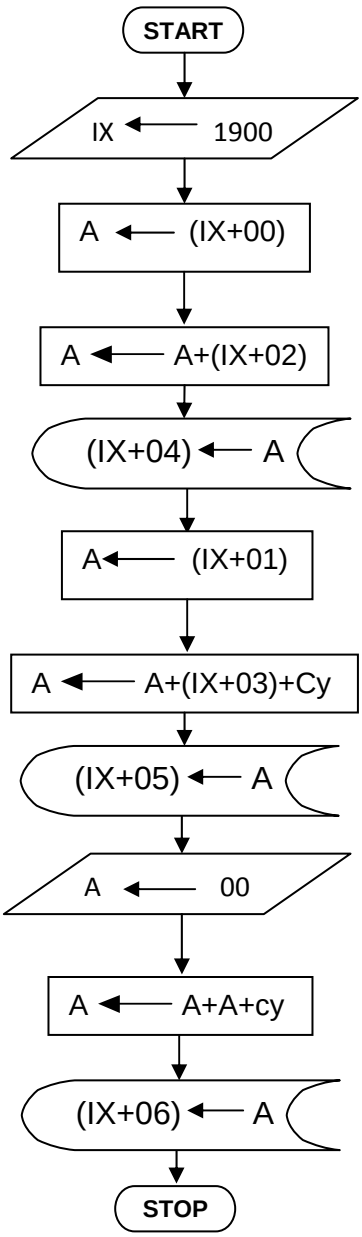
Gambar 6.27. Penjumlahan bilangan 2 byte

Dari Gambar 6.26. dapat dikembangkan algoritma sebagai berikut.

Algoritma :

1. *Ambil data yang ada pada alamat 1900 dan simpan di Reg. A*
2. *Jumlahkan data yang ada pada Reg. A dengan data yang ada pada alamat 1902*
3. *Simpan hasil penjumlahan yang ada di Reg. A pada alamat 1904*
4. *Ambil data yang ada pada alamat 1901 dan simpan di Reg. A*
5. *Jumlahkan data yang ada pada Reg. A dengan data yang ada pada alamat 1903 dan sertakan bit carry flag*
6. *Simpan hasil penjumlahan yang ada di Reg. A pada alamat 1905*
7. *Ambil bit carry flag dan simpan pada alamat 1906.*

Kemudian flowchart permasalahan penjumlahan dua byte bilangan biner dapat dibuat seperti Gambar 6.28.



Gambar 6.28. Flowchart penjumlahan 2 byte bilangan biner

Dari flowchart Gambar 6.28. langkah pertama register IX diisi data immediate 1900 sebagai basis nilai alamat tempat simpan data. Pada step ke dua data pada alamat 1900 ditransfer ke register A menggunakan instruksi LDA, (IX+00). Data dari alamat 1900 kemudian dijumlahkan dengan data yang ada pada alamat 1902 dengan instruksi ADD A, (IX+02). Hasil penjumlahan byte pertama masih tercatat di register A, maka selanjutnya data register A disimpan ke memori alamat 1904 dengan instruksi LD (IX+04), A. Sampai disini penjumlahan byte pertama telah selesai.

Penjumlahan byte berikutnya dilakukan dengan mengambil data byte kedua dari alamat 1901 ke register A dengan instruksi LDA, (IX+01). Data byte kedua yang ada di register A dijumlahkan dengan data memori alamat 1903 dengan menyertakan carry. Instruksi yang digunakan ADC A, (IX+03). Instruksi ADC harus digunakan buka ADD untuk mengambil dan menyertakan carry penjumlahan byte pertama. Hasil penjumlahan byte kedua kemudian disimpan ke memori alamat 1905 dengan instruksi LD (IX+05),A. Penjumlahan byte kedua memberi kemungkinan adanya carry. Untuk mengambil nilai carry pertama register A dibuat bernilai 0, lalu dilakukan penjumlahan A=A+A+Cy. Karena nilai A =0 maka hasil penjumlahan A=Cy. Kemudian diteruskan dengan menyimpan nilai register A ke memori alamat 1906 menggunakan instruksi LD (IX+06), A. Algoritma Gambar 6.26. dan Flowchart Gambar 6.27. dapat digunakan sebagai dasar penulisan program. Program penjumlahan 2 byte bilangan biner adalah seperti Gambar 6.29.

Program :

Addr	Kode Operasi	No.	Label	Assembly	Ket
1800	DD 21 00 19	1		LD IX , 1900	
1804	DD 7E 00	2		LD A , (IX+00)	
1807	DD 86 02	3		ADD A,(IX+02)	
180A	DD 77 04	4		LD (IX+04) , A	
180D	DD 7E 01	5		LD A, (IX+01)	
1810	DD 8E 03	6		ADC A,(IX+03)	
1813	DD 77 05	7		LD (IX+05) , A	
1816	3E 00	8		LD A , 00	
1818	8F	9		ADC A , A	
1819	DD 77 06	10		LD (IX+06), A	
181C	FF	11		RST 38	

Gambar 6.29. Program penjumlahan 2 byte bilangan biner

Program Gambar 6.28 bisa diuji apakah memberi hasil yang benar. Dengan menggunakan data seperti Tabel 6.1. berikut isilah hasil penjumlahan dari tiga kasus masing-masing.

Tabel 6.1. Pengujian program penjumlahan 2 byte

Data 1		Data 2		Data 1 + Data 2		
[1900]	[1901]	[1902]	[1903]	[1904]	[1905]	[1906]
10	00	03	02			
06	05	08	07			
8F	9F	F5	AF			

Program Gambar 6.28. adalah program penjumlahan bilangan biner. Untuk membuat program tersebut bekerja sebagai program penjumlahan bilangan desimal maka perlu di tambahkan satu jenis perintah koreksi DAA pada langkah nomor 4 dan langkah nomor 8 seperti Gambar 6.30

Addr	Kode Operasi	No.	Label	Assembly	Ket.
1800	DD 21 00 19	1		LD IX , 1900	
1804	DD 7E 00	2		LD A, (IX+00)	
1807	DD 86 02	3		ADDA, (IX+02)	
180A	27	4		DAA	Koreksi
180B	DD 77 04	5		LD (IX+04) , A	
180E	DD 7E 01	6		LD A , (IX+01)	
1811	DD 8E 03	7		ADC A, (IX+03)	
1814	27	8		DAA	Koreksi
1815	DD 77 05	9		LD (IX+05) , A	
1818	3E 00	10		LD A , 00	
181A	8F	11		ADC A , A	
181B	DD 77 06	12		LD (IX+06) , A	
181E	FF	13		RST 38	

Gambar 6.30. Program penjumlahan 2 byte bilangan desimal

Program Gambar 6.29 diuji menggunakan data seperti Tabel 6.2.

Tabel 6.2. Pengujian program penjumlahan 2 byte

Data 1		Data 2		Data 1 + Data 2		
[1900]	[1901]	[1902]	[1903]	[1904]	[1905]	[1906]
10	20	30	40			
15	25	50	70			
40	15	25	55			

Kasus kedua adalah masalah pengurangan satu byta data. Berikut disajikan Algoritma Program, Flow Chart dan Program untuk pengurangan 1 byte bilangan biner yang masing-masing beralamat di 1900H, 1901H dan hasilnya disimpan pada alamat 1902. Program di tulis pada alamat awal program 1800H.

Penyelesaian masalah pengurangan satu byte data dapat diformulasikan seperti Gambar 6.31.

[1900]
[1901]
---------- -
[1902]

Gambar 6.31. Formulasi pengurangan satu byte data.

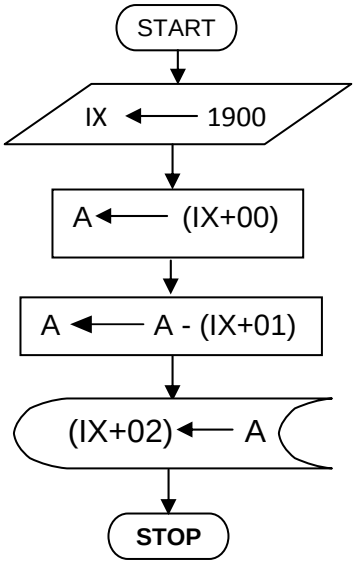
Dari Gambar 6.30 dapat dikembangkan algoritma sebagai berikut.

Algoritma :

1. *Ambil data yang ada pada alamat 1900 dan simpan di Reg. A*
2. *Kurangkan data yang ada pada Reg. A dengan data yang ada pada alamat 1901*
3. *Simpan hasil pengurangan yang ada di Reg. A pada alamat 1902*

Ada tiga langkah dalam algoritma yang harus dijalankan. Untuk lebih memperjalan alur algoritma ini selanjutnya dikembangkan flowchart seperti Gambar 6.32.

Flow Chart



Gambar 6.32. Flowchart pengurangan satu byte data

Flowchart Gambar 6.32. menunjukkan proses aliran program diawali dengan start kemudian menetapkan register IX sebagai pencatat alamat dengan nilai dasar 1900. Data pada alamat 1900 ditransfer ke register A dengan instruksi transfer data 8 bit LD A,(IX+00). Langkah berikutnya adalah mengurangkan data yang ada di register A dengan data pada alamat 1901. Instruksi yang dipakai adalah SUB ,(IX+01). Hasil pengurangan masih ada di register A dan selanjutnya ditransfer ke memori alamat 1902 dengan instruksi LD (IX+02),A. Pengurangan ini memberi dua kemungkinan hasil yaitu nilai positif jika bilangan yang dikurangi lebih besar dari bilangan pengurangnya dan akan bernilai negatif jika bilangan yang dikurangi lebih kecil dari bilangan pengurangnya. Dan nilai hasil pengurangan akan sama dengan nol jika bilangan yang dikurangi sama dengan bilangan pengurang. Jika nilai hasil pengurangnya negatif maka nilai riilnya didapat dengan mengkonversi menjadi komplemen dua. Program lengkap pengurangan bilangan 1 byte ada pada Gambar 6.33.

Program :

Addr	Kode Operasi	No.	Label	Assembly	Ket
1800	DD 21 00 19	1		LD IX , 1900	
1804	DD 7E 00	2		LD A , (IX+00)	
1807	DD 96 05	3		SUB , (IX+01)	
180A	DD 77 02	4		LD (IX+02) , A	
180D	FF	5		RST 38	

Gambar 6.33. Program pengurangan satu byte data bilangan biner

Program Gambar 6.33 di atas bisa diuji menggunakan data seperti Tabel 6.3. Coba diisi hasil pengurangan pada Tabel 6.3.

Tabel 6.3. Pengujian pengurangan satu byte bilangan biner

[1900]	[1901]	[1902]
F5	2D	
E7	D5	
8F	F5	

Program diatas adalah program pengurangan bilangan biner. Untuk membuat program tersebut bekerja sebagai program pengurangan bilangan desimal

maka perlu di tambahkan satu perintah koreksi DAA pada langkah 4 seperti Gambar 6.34.

Addr	Kode Operasi	No	Label	Assembly	Ket
1800	DD 21 00 19	1		LD IX , 1900	
1804	DD 7E 00	2		LD A, (IX+00)	
1807	DD 96 05	3		SUB, (IX+01)	
180A	27	4		DAA	
180B	DD 77 02	5		LD (IX+02) , A	
180E	FF	6		RST 38	

Gambar 6.34. Program pengurangan satu byte data bilangan desimal

Pengujian Program :

Jika program Gambar 6.34. dieksekusi coba isi hasil ppengurangan pada Tabel 6.4. berikut .

Tabel 6.4. Pengujian pengurangan satu byte bilangan desimal

[1900]	[1901]	[1902]
80	40	
90	25	
55	15	
15	55	

Program pengurangan dapat dibuat menjadi program penjumlahan dengan menegatifkan bilangan pengurangnya. Program berikut menghasilkan keluaran yang sama dengan program Gambar 6.34.

Addr	Kode Operasi	No.	Label	Assembly	Ket
1800	DD 21 00 19	1		LD IX ,1900	
1804	DD 7E 01	2		LD A, (IX+01)	
1807	ED 44	3		NEG	
1809	27	4		DAA	
180A	47	5		LD B , A	
180B	DD 7E 00	6		LD A , (IX+00)	
180E	80	7		ADD A , B	
180F	27	8		DAA	
1810	DD77 02	9		LD (IX+02) , A	
1813	FF	10		RST 38	

Gambar 6.35. Program pengurangan satu byte data bilangan desimal

Program Gambar 6.35. memodifikasi program Gambar 6.34. Register IX diisi data immediate 1900 sebagai basis pencatat alamat memori. Untuk merubah pengurangan menjadi penjumlahan dilakukan proses A-B = A+(-B). Berdasarkan kasus ini maka pertama bilangan pengurang diambil terlebih dahulu dari memori alamat 1901 kemudian dinegatifkan denganperintah NEG. Selanjutnya bilangan yang dikurangi ditransfer ke register A lalu dijumlahkan dengan bilangan pengurang yang sebelumnya sudah dinegatifkan. Agar diperoleh hasil yang benar maka koreksi DAA harus disertakan setelah instruksi penambahan dilakukan.

6. Instruksi Putar & Geser

Instruksi putar dan geser sangat efektif sekali digunakan untuk pengolahan bit dalam suatu register atau memori. Perintah-perintah yang digunakan adalah :

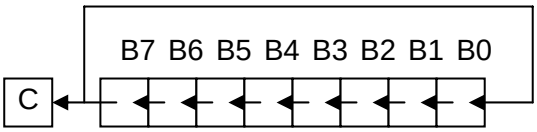
- RLC A : Rotate Left Circular Accumulator
- RL A : Rotate Left Accumulator
- RRC A : Rotate Right Circular Accumulator
- RR A : Rotate Right Accumulator
- RLC r : Rotate Left Circular r (salah satu register utama 8 bit)
- RL s : Rotate Left s (salah satu register utama 8 bit, memori yang alamatnya dicatat (HL), (IX+d) , (IY+d)
- RRC r : Rotate Right Circular r (salah satu register utama 8 bit)
- RR s : Rotate Right s (salah satu register utama 8 bit, memori yang alamatnya dicatat (HL), (IX+d) , (IY+d)
- SLA s : Shift Left Arithmetic s (salah satu register utama 8 bit, memori yang alamatnya dicatat (HL), (IX+d) , (IY+d)
- SRA s : Shift Right Arithmetic s (salah satu register utama 8 bit, memori yang alamatnya dicatat (HL), (IX+d) , (IY+d)
- RLD : Rotate Digit Left diantara akumulator dengan lokasi memori yang dicatat oleh (HL)
- RRD : Rotate Digit Right diantara akumulator dengan lokasi memori yang dicatat oleh (HL)

6.1. Rotate Left Circular (RLC)

Rotate left circular bekerja memutar bit dalam satu byte data ke kiri dengan memasukkan bit B7 ke Carry Flag. Dalam hal ini berlaku proses:

(Bn+1) ← (Bn) dimana n = 0 s /d 6
(B0) ← (B7)
(CY) ← (B7)

Secara diagram rotate left circular dapat digambarkan seperti Gambar 6.36.



Gambar 6.36. Diagram rotate left circular

RLC bekerja memutar bit B0 ke B1, B1 ke B2, B2 ke B3, B3 ke B4, B4 ke B5, B5 ke B6, B6 ke B7 dan B7 ke B0 disamping juga B7 ke Cy.

Contoh Instruksi RLC adalah:

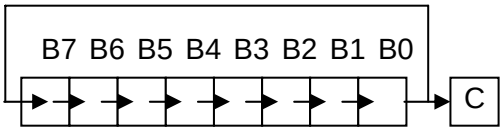
```
RLC A
RLC (HL)
RLCA
RLC (IX+d)
RLC B
RLC (IY+d)
RLC C
RLC D
RLC E
RLC H
RLC L
```

6.2. Rotate Right Circular (RRC)

Rotate right circular bekerja memutar bit dari byte data ke kanan dengan memasukkan bit B0 ke Carry Flag. Dalam hal ini berlaku proses:

(Bn) ← (Bn+1) dimana n = 0 s /d 6
(B7) ← (B0)
(CY) ← (B0)

Secara diagram rotate right circular dapat digambarkan seperti Gambar 6.37.



Gambar 6.37. Diagram rotate right circular

RRC bekerja memutar bit B7 ke B6, B6 ke B5, B5 ke B4, B4 ke B3, B3 ke B2, B2 ke B1, B1 ke B0 dan B0 ke B7disamping juga B0 ke Cy.

Contoh Instruksi RRC adalah:

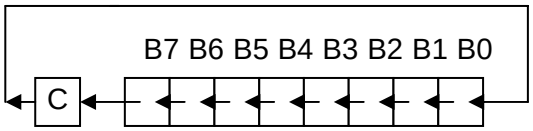
RRC A
RRC (HL)
RRCA
RRC (IX+d)
RRC B
RRC (IY+d)
RRC C
RRC D
RRC E
RRC H
RRC L

6.3. Rotate Left (RL)

Rotete left bekerja memutar bit dari byte data ke kiri dengan memasukkan bit B7 ke Carry Flag dan Carry Flag ke B0. Dalam hal ini berlaku proses:

(Bn+1) ← (Bn) dimana n = 0 s /d 6
(B0) ← (CY)
(CY) ← (B7)

Secara diagram rotate left dapat digambarkan seperti Gambar 6.38.



Gambar 6.38. Diagram rotate left

RL bekerja memutar bit B0 ke B1, B1 ke B2, B2 ke B3, B3 ke B4, B4 ke B5, B5 ke B6, B6 ke B7, B7 ke Cy, dan Cy ke B0.

Contoh Instruksi RL adalah:

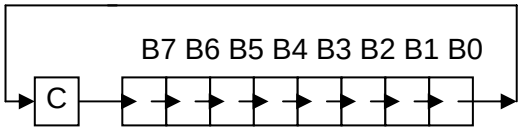
RL A
RL (HL)
RLA
RL (IX+d)
RL B
RL (IY+d)
RL C
RL D
RL E
RL H
RL L

6.4. Rotate Right (RR)

Memutar bit byte data ke kanan dengan memasukkan bit B0 ke Carry Flag dan Carry Flag ke B7. Dalam hal ini berlaku proses:

(Bn) ← (Bn+1) dimana n = 0 s /d 6
(CY) ← (B0)
(B7) ← (CY)

Secara diagram rotate right circular dapat digambarkan seperti Gambar 6.39.



Gambar 6.39. Diagram rotate right

RR bekerja memutar bit B7 ke B6, B6 ke B5, B5 ke B4, B4 ke B3, B3 ke B2, B2 ke B1, B1 ke B0, B0 ke Cy dan Cy ke B7.

Contoh Instruksi RR:

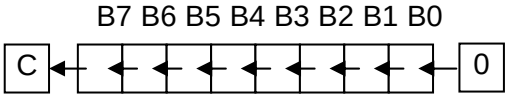
RR A
RR (HL)
RRA
RR (IX+d)
RR B
RR (IY+d)
RR C
RR D
RR E
RR H
RR L

5.5. Shift Left Arithmetic

SLA adalah perintah menggeser bit data ke kiri dengan memasukkan bit B7 ke Carry Flag. Dalam hal ini berlaku proses:

(Bn+1) ← (Bn) dimana n = 0 s /d 6
(CY) ← (B7)
(B0) ← 0

Secara diagram rotate right circular dapat digambarkan seperti Gambar 6.40.



Gambar 6.40. Diagram shift left arithmetic

SLA bekerja memutar data 0 ke bit B0, B0 ke B1, B1 ke B2, B2ke B3, B3 ke B4, B4 ke B5, B5 ke B6, B6 ke B7, B7 ke Cy.

Contoh Instruksi SLA:

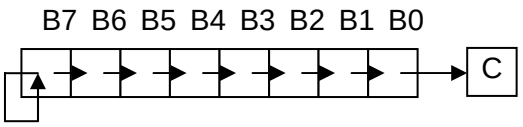
SL A
SL (HL)
SL B
SL (IX+d)
SL C
SL (IY+d)
SL D
SL E
SL H
SL L

6.6. Shift Right Arithmetic (SRA)

SRA bekerja menggeser bit dari byte data ke kanan dengan memasukkan bit B0 ke Carry Flag. Dalam hal ini berlaku proses:

(Bn) ← (Bn+1) dimana n = 0 s /d 6
(CY) ← (B0)
(B7) ← (B7)

Secara diagram SRA dapat digambarkan seperti Gambar 6.41.



Gambar 6.41. Diagram SRA

SLA bekerja memutar data B7 ke B6, B6 ke B5, B5 ke B4, B4 ke B3, B3 ke B2, B2 ke B1, B1 ke B1, B0 ke Cy dan B7 tetap B7.

Contoh Instruksi SRA:

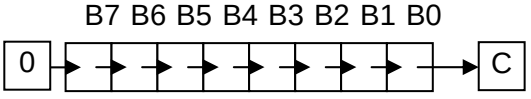
SR A
SR (HL)
SR B
SR (IX+d)
SR C
SR (IY+d)
SR D
SR E
SR H
SR L

6.7. Shift Right Logical (SRL)

SRL menggeser bit data ke kanan dengan memasukkan bit B0 ke Carry Flag dan B7 diisi 0. Dalam hal ini berlaku proses:

(Bn) ← (Bn+1) dimana n = 0 s /d 6
(CY) ← (B0)
(B7) ← 0

Secara diagram SRL dapat digambarkan seperti Gambar 6.42.



Gambar 6.42. Diagram SRL

Contoh Instruksi:

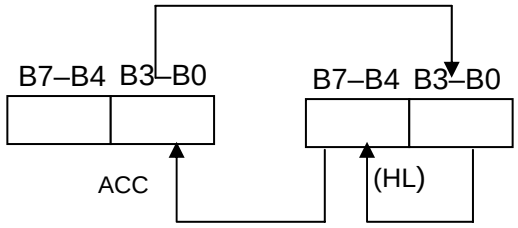
SRL A
SRL (HL)
SRL B
SRL (IX+d)
SRL C
SRL (IY+d)
SRL D
SRL E
SRL H
SRL L

6.8. Rotate Left Digit (RLD)

Memutar nible data di memori yang alamatnya dicatat oleh register HL ke kiri melibatkan register A. Dalam hal ini berlaku proses:

Pada (HL)
(B7- B4) ← (B3 - B0)
Pada ACC
(B3- B0) ← (B7– B4) pada (HL)

Secara diagram RLD dapat digambarkan seperti Gambar 6.43.



Gambar 6.43. Diagram RLD

Instruksi RLD

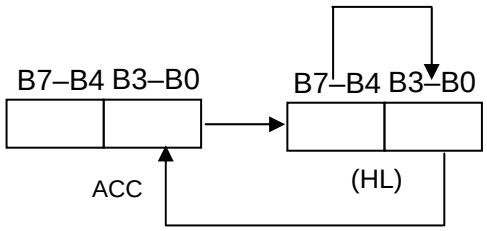
6.9. Rotate Right Digit (RRD)

Memutar nible data di memori yang alamatnya dicatat oleh register HL ke kanan melibatkan register A. Dalam hal ini berlaku proses:

Pada (HL)
(B3- B0) ← (B7 – B4)

pada ACC
(B7- B4) pada (HL) ← (B3 – B0)

Secara diagram RRD dapat digambarkan seperti Gambar 6.44.



Gambar 6.44. Diagram RRD

Instruksi **RRD**

7. Instruksi Manipulasi Bit

Manipulasi bit berkaitan dengan set bit, reset bit, dan test bit dari sebuah register. Instruksi in berkaitan dengan penggunaan register A, B, C, D, E, H, L, dan memori yang dialamati oleh register HL, IX, dan IY. Gambar 6.45. menunjukkan contoh perintah manipulasi bit.

Operasi	Assembly	Operasi	Ket.
1 Bit	BIT 0 , A	Z ← A0*	Mempengaruhi Flag Z Z = 1 jika bit yang ditunjuk 0 Z = 0 jika bit yang ditunjuk 1
	BIT 1 , B	Z ← B1*	
	BIT 7 , A	Z ← A7*	
	BIT 4 , D	Z ← D4*	
	BIT 1, (HL)	Z←(HL)1*	
	BIT 3 ,(IX+d)	Z←(IX+d)3*	
	BIT 7 , (IY+d)	Z←(IY+d)7*	
	SET 0, A	A0 ← 1	
	SET 7 , B	B7 ← 1	
	SET 4 , (IX+d)	(IX+d)4← 1	
	SET 5 , (IY+d)	(IY-d)5 ← 1	
	SET 6, (HL)	(HL)6 ← 1	
	RES 0, A	A0 ← 0	
	RES 7 , B	B7 ← 0	
	RES 4, (IX+d)	(IX+d)4 ← 0	
	RES 5, (IY+d)	(IY-d)5 ← 0	
	RES 6, (HL)	(HL)6 ← 0	

Gambar 6.45. Contoh instruksi manipulasi bit

Dari Gambar 6.45. sebagian bisa dijabarkan dalam kelompok test bit, set, dan reset. Instruksi BIT 0, A adalah instruksi test bit 0 dari data 8 bit yang ada di register A. Instruksi ini dapat digunakan untuk menguji apakah bit 0 dari register A berlogika 0 atau 1. Jika Zerro flag sama dengan 0 berarti data BIT 0 pada register A adalah 1 dan jika Z=1 berarti bit 0 dari data 8 bit register A adalah 0. Test bit bisa juga untuk bit yang lain dari byte data di register A atau register yang termasuk data yang ada di memori.

Untuk membuat bit data dari satu byte data berlogika 1 atau 0 digunakan instruksi SET atau RES. Instruksi ini dapat ke register, atau memori.

8. Instruksi JUMP

Dalam mikroprosesor Zilog Z-80 CPU instruksi-instruksi pencabangan mengguna kan instruksi **JUMP**. Instruksi JUMP membuat mikroprosesor menjadi perangkat yang sangat ampuh. Instruksi JUMP dapat dikategorikan menjadi empat kategori yaitu :

- **JUMP bersyarat**
- **JUMP tanpa syarat**
- **JUMP absolut**
- **JUMP relatif**

8.1. JUMP Bersyarat

Jump bersyarat adalah jenis instruksi Jump yang bekerja melakukan lompatan atau kontinyu/tidak melompat berdasarkan syarat yang diberikan. Mnemonik untuk lompatan bersyarat ada tiga yaitu :

1. **JP cc** : Lompatan absolut bersyarat adalah lompatan yang langsung menunjuk alamat sasaran dengan data alamat 16 bit.
2. **JR cc** : Lompatan Relatif bersyarat adalah lompatan yang penunjukan alamatnya bernilai relatif terhadap alamat posisi saat melompat.
3. **DJNZ** : Lompatan Relatif Khusus terhadap register B adalah lompatan yang penunjukan alamatnya bernilai relatif terhadap alamat posisi saat melompat.

Bentuk perintah Jump bersyarat adalah sebagai berikut:

- Lompatan Absolut bersyarat:
  JP cc, nn :
  Jika kondisi syarat cc terpenuhi maka PC ← nn artinya melompat
  Jika syarat cc tidak terpenuhi maka kontinyu

- Lompatan Relatif bersyarat :
  JR cc, n:
  Jika kondisi syarat cc terpenuhi maka PC ← PC + e artinya melompat.
  Jika syarat cc tidak terpenuhi maka kontinyu

- Lompatan Relatif bersyarat Khusus : **DJNZ** :
  B ← B - 1
  Jika B = 0 kontinyu dan
  jika B <> 0 maka
  PC ← PC + e

Syarat yang dimaksud untuk setiap perintah Jump terkait dengan kondisi bit status Flag dari satu step perintah sebelumnya. Ada 6 kemungkinan syarat yang dapat diberikan terkait dengan bit status flag. Kedelapan syarat itu dalam mikroprosesor dicatat dalam sebuah register yang disebut dengan register Flag. Untuk mikroprosesor Z-80 CPU susunan status dari register flag seperti Gambar 6.46

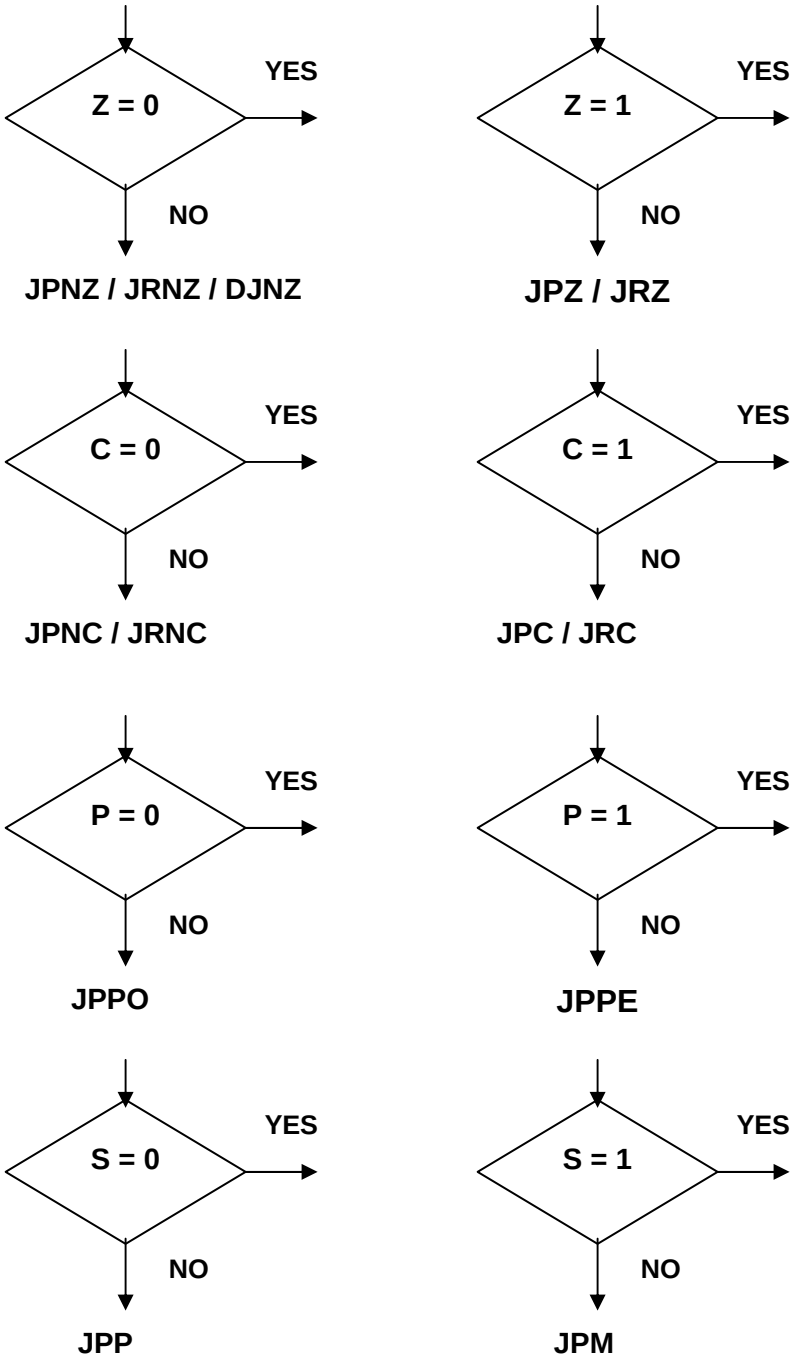
S	Z	X	H	X	P/V	N	C

Gambar 6.46 Susunan Register Flag Z-80 CPU

Makna masing-masing bit dari tanda status flag pada register F adalah sebagai berikut :

S (Sign)	= 1	Menunjukkan hasil operasi aritmetika/logika sebelumnya bertanda negatif (b7 = 1)
	= 0	Menunjukkan hasil operasi aritmetika/logika sebelumnya bertanda positif (b7 = 0)
Z (Zerro)	= 1	Menunjukkan hasil operasi aritmetika/logika sebelumnya bernilai nol
	= 0	Menunjukkan hasil operasi aritmetika/logika sebelumnya bernilai tidak nol
H (Half-	**=** 1	Jika ada carry dari bit B3 ke bit B4
Carry)	**=** 0	Jika tidak ada carry dari bit B3 ke bit B4
P/V(Parity/	= 1	Jika hasil operasi
Overflow)		aritmetika/logika sebelumnya menunjukkan paritas genap atau terjadi Overflow
	= 0	Jika hasil operasi aritmetika/logika sebelumnya menunjukkan paritas ganjil atau tidak terjadi Overflow
N	= 1	Operasi sebelumnya
(Non Carry)		adalah operasi SUBTRACK/Penguran gan
	= 0	Operasi sebelumnya adalah operasi bukan SUBTRACK/Penguran gan
C (Carry)	= 1	Jika operasi sebelumnya menghasilkan Carry atau borrow
	= 0	Jika operasi sebelumnya tidak menghasilkan Carry atau borrow
X	**: tidak digunakan**	

Pengambilan keputusan melompat atau kontinyu sebuah proses program terkait langsung dengan status flag register tersebut. Untuk memudahkan memahami terjadinya lompatan untuk setiap persyaratan pada setiap perintah dapat digambarkan seperti Gambar 6.47. berikut.



Gambar 6. 47. Bentuk-bentuk pengambilan keputusan JUMP

Dari Gambar 6.46 diatas:

- **JP NZ, nn** : mengandung makna Jump absolut if Not Zerro yaitu jika hasil operasi ALU sebelumnya tidak bernilai 0 atau nilai flag Z = 0 maka keputusan melompat dilaksanakan ke lokasi alamat absolut nn dimana nn adalah alamat 16 bit. Sebaliknya jika Z=1 maka program counter akan diteruskan atau kontinyu naik satu step tanpa lompatan.
- **JR NZ, n** mengandung makna Jump relatif if not zerro yaitu jika hasil operasi sebelumnya tidak bernilai 0 atau nilai flag Z = 0 maka keputusan melompat dilaksanakan ke lokasi alamat relatif n dimana n adalah nilai relatif alamat yang dituju. Dan jika Z = 1 maka step program akan kontinyu ke satu langkah berikutnya.
- **DJNZ, n** mengandung makna Decrement B Jump if Not zerro yaitu jika hasil operasi pengurangan nilai B tidak sama dengan 0 atau nilai flag Z = 0 maka keputusan melompat dilaksanakan ke lokasi alamat relatif n dimana n adalah nilai relatif alamat yang dituju. Dan jika nilai B sama dengan 0 atau Z = 1 maka step program akan kontinyu ke satu langkah berikutnya.
- **JP Z, nn** : mengandung makna Jump absolut if Zerro yaitu jika hasil operasi ALU sebelumnya bernilai 0 atau nilai flag Z = 1 maka keputusan melompat dilaksanakan ke lokasi alamat absolut nn dimana nn adalah alamat 16 bit. Sebaliknya jika Z=0 maka program counter akan diteruskan atau kontinyu naik satu step tanpa lompatan.
- **JR Z, n** mengandung makna Jump relatif if zerro yaitu jika hasil operasi sebelumnya bernilai 0 atau nilai flag Z = 1 maka keputusan melompat dilaksanakan ke lokasi alamat relatif n dimana n adalah nilai relatif alamat yang dituju. Dan jika Z = 0 maka step program akan kontinyu ke satu langkah berikutnya.
- **JP NC, nn** : mengandung makna Jump absolut if Not Carry yaitu jika hasil operasi ALU sebelumnya tidak ada Carry atau nilai flag C = 0 maka keputusan melompat dilaksanakan ke lokasi alamat absolut nn dimana nn adalah alamat 16 bit. Sebaliknya jika C=1 maka program counter akan diteruskan atau kontinyu naik satu step tanpa lompatan.
- **JR NC, n** mengandung makna Jump relatif if Not Carry yaitu jika hasil operasi sebelumnya tidak ada Carry atau nilai flag C = 0 maka

keputusan melompat dilaksanakan ke lokasi alamat relatif n dimana n adalah nilai relatif alamat yang dituju. Dan jika C = 1 maka step program akan kontinyu ke satu langkah berikutnya.

- **JP C, nn** : mengandung makna Jump absolut if Carry yaitu jika hasil operasi ALU sebelumnya ada Carry atau nilai flag C = 1 maka keputusan melompat dilaksanakan ke lokasi alamat absolut nn dimana nn adalah alamat 16 bit. Sebaliknya jika C=0 maka program counter akan diteruskan atau kontinyu naik satu step tanpa lompatan.
- **JR C, n** mengandung makna Jump relatif if Carry yaitu jika hasil operasi sebelumnya tidak ada Carry atau nilai flag C = 1 maka keputusan melompat dilaksanakan ke lokasi alamat relatif n dimana n adalah nilai relatif alamat yang dituju. Dan jika C = 0 maka step program akan kontinyu ke satu langkah berikutnya.
- **JPPO, nn** : mengandung makna Jump absolut if Parity Odd (ganjil) yaitu jika hasil operasi ALU sebelumnya paritasnya ganjil atau nilai flag P = 0 maka keputusan melompat dilaksanakan ke lokasi alamat absolut nn dimana nn adalah alamat 16 bit. Sebaliknya jika P=1 maka program counter akan diteruskan atau kontinyu naik satu step tanpa lompatan.
- **JPPE, nn** : mengandung makna Jump absolut if Parity Even (genap) yaitu jika hasil operasi ALU sebelumnya paritasnya ganjil atau nilai flag P = 1 maka keputusan melompat dilaksanakan ke lokasi alamat absolut nn dimana nn adalah alamat 16 bit. Sebaliknya jika P=0 maka program counter akan diteruskan atau kontinyu naik satu step tanpa lompatan.
- **JPP, nn** : mengandung makna Jump absolut if Pluss yaitu jika hasil operasi ALU sebelumnya nilai plus atau bit B7=0 atau nilai flag S = 0 maka keputusan melompat dilaksanakan ke lokasi alamat absolut nn dimana nn adalah alamat 16 bit. Sebaliknya jika S=1 maka program counter akan diteruskan atau kontinyu naik satu step tanpa lompatan.
- **JPM, nn** : mengandung makna Jump absolut if Minus yaitu jika hasil operasi ALU sebelumnya bernilai minus atau bit B7=1 atau nilai flag S = 1 maka keputusan melompat dilaksanakan ke lokasi alamat absolut nn dimana nn adalah alamat 16 bit. Sebaliknya jika S=0

maka program counter akan diteruskan atau kontinyu naik satu step tanpa lompatan.

8.2. JUMP Tanpa Syarat

Jump tanpa syarat adalah jenis instruksi Jump yang bekerja melakukan lompatan atau kontinyu tanpa adanya syarat yang diberikan. Mnemonik untuk lompatan tanpa syarat ada dua yaitu :

1. **JP** : Lompatan absolut tanpa syarat adalah lompatan yang langsung menunjuk alamat sasaran dengan data alamat 16 bit.
2. **JR** : Lompatan Relatif tanpa syarat adalah lompatan yang penunjukan alamatnya bernilai relatif terhadap alamat posisi saat melompat.

8.2.1. JUMP Absolut tanpa syarat :

JP nn : PC ← nn

Instruksi ini memasukkan alamat memori nn ke register PC (Program Counter), sehingga meikroprosesor akan menjalankan instruksi yang ada pada lokasi alamat nn.

8.2.2. Lompatan Relatif tanpa syarat :

JR e : PC ← PC + e

e adalah bilangan bertanda yang bernilai positif jika melompat maju ke alamat berikutnya dan bernilai negatif jika melompat ke belakang ke alamat sebelumnya.

Nilai relatif lompatan dapat dihitung dengan rumus :

1. Jika melompatnya maju ke alamat di atasnya :
   e = d - (S + 02)
2. Jika melompatnya mundur ke alamat sebelumnya :
   e = (S + 02) – d

lalu dikomplemen duakan

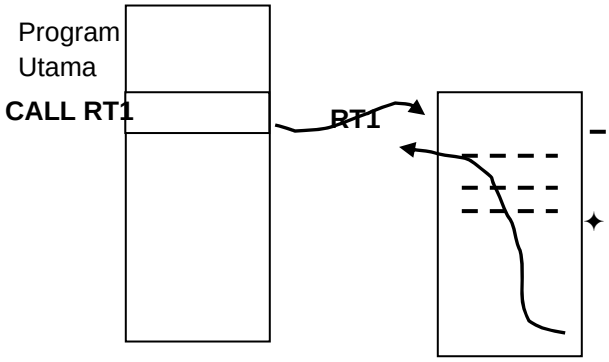
dimana d = alamat tujuan dan S = alamat asal perintah Jump

Disamping Jump bersyarat masih ada tiga jenis Jump lainnya yaitu Jump Absolut berbasis register HL, IX, dan IY dengan mnemonic :

JP (HL) : PC ← HL
JP (IX) : PC ← IX
JP (IY) : PC ← IY

9. Instruksi Call dan Return

Instruksi CALL dan RET digunakan untuk membangun dan menjalankan sub routine program. Sub routine program adalah routine sebagai penggalan program aplikasi yang dapat dipanggil secara berulang-ulang dari program utama. Untuk pemanggilan sub routine digunakan perintah CALL. Sedangkan pengaturan untuk kembali lagi ke program utama digunakan perintah RET. Pembentukan layanan sub routine dapat digambarkan seperti Gambar 6.48.



Gambar 6.48 . Diagram proses CALL dan Return

Instruksi CALL dan RET ada dua jenis yaitu :

1. **CALL tanpa Syarat**
2. **CALL bersyarat**
3. **RET tanpa Syarat**
3. **RET bersyarat**

✦ **CALL tanpa Syarat**

Instruksi CALL tanpa syarat adalah perintah menjalankan suatu sub routine program dengan menunjuk alamat absolut dari program sub routine program tersebut. Perintah Call tanpa syarat dinyatakan dengan perintah :

CALL nn

Pada saat ini terjadi suatu proses :

(SP-1) ← PCH
(SP-2) ← PCL
PC ← nn

Nilai dari program counter pada program utama dimasukkan ke stack semacam perintah PUSH, dan PC diisi nilai alamat awal nn dari sub routine yang di CALL.

✦ **CALL bersyarat**

Instruksi CALL bersyarat adalah perintah menjalankan suatu sub routine program dengan menunjuk alamat absolut dari program sub routine program tersebut jika syarat yang ditetapkan terpenuhi. Perintah Call bersyarat dinyatakan dengan perintah :

CALL cc, nn

Pada saat ini terjadi suatu proses : jika syarat atau kondisi terpenuhi maka

(SP-1) ← PCH
(SP-2) ← PCL
SP ← SP-2
PC ← nn

Nilai dari program counter pada program utama dimasukkan ke stack semacam perintah PUSH, dan PC diisi nilai alamat awal nn dari sub routine yang di CALL.

Jika syarat atau kondisi tidak terpenuhi maka program counter tidak mencabang atau kontinyu keperintah berikutnya.

Perintah CALL bersyarat adalah :

CALL C, nn : Jika ada carry flag atau Cy = 1 maka PC diisi nn untuk keadaan lain PC kontinyu.

CALL M, nn: Jika ada Sign flag atau S = 1 maka PC diisi nn untuk keadaan lain PC kontinyu

CALL NC,**nn** : Jika tidak ada carry flag atau Cy = 0 maka PC diisi nn untuk keadaan lain PC kontinyu.

CALL NZ, nn : Jika hasil operasi ALU sebelumnya tidak sama dengan nol atau Z = 0 maka PC diisi nn untuk keadaan lain PC kontinyu

CALL Z, nn : Jika hasil operasi ALU sebelumnya sama dengan nol atau Z = 1 maka PC diisi nn untuk keadaan lain PC kontinyu

CALL P, nn : Jika tidak ada Sign flag atau S = 0 maka PC diisi nn untuk keadaan lain PC kontinyu

CALL PE, nn : Jika hasil operasi ALU sebelumnya paritasnya genap atau nilai flag P = 1 maka PC diisi nn untuk keadaan lain PC kontinyu

CALL PO, nn : Jika hasil operasi ALU sebelumnya paritasnya ganjil atau nilai flag P = 0 maka maka PC diisi nn untuk keadaan lain PC kontinyu

✦ **RETURN tanpa Syarat**

Perintah RETURN tanpa syarat digunakan untuk mengembalikan program counter ke program utama setelah menyelesaikan suatu sub routine. Perintah RETURN tanpa Syarat adalah:

RET

Pada saat ini terjadi proses :

PCL ← (SP)
PCH ← (SP+1)
SP ← SP+2

✦ **RETURN Bersyarat**

Perintah RETURN bersyarat digunakan untuk mengembalikan program counter ke program utama setelah menyelesaikan suatu sub routine jika syarat yang ditetapkan terpenuhi. Jika syarat yang ditetapkan tidak terpenuhi maka PC kontinyu di sub routine.

Perintah RETURN tanpa Syarat adalah:

RET cc

Pada saat ini terjadi proses :

Jika syarat terpenuhi :

PCL ← (SP)
PCH ← (SP+1)
SP ← SP+2

Dan jika syarat tidak terpenuhi maka : PC kontinyu ke instruksi berikutnya di Sub routine.

Perintah RET bersyarat adalah :

RET C : Jika ada carry flag atau Cy = 1 maka PC diisi nilai dari stack, kembali ke program utama. Untuk keadaan lain PC kontinyu.

RET M : Jika ada Sign flag atau S = 1 maka PC diisi nilai dari stack, kembali ke program utama. Untuk keadaan lain PC kontinyu

RET NC: Jika tidak ada carry flag atau Cy = 0 maka PC diisi nilai dari stack, kembali ke program utama. Untuk keadaan lain PC kontinyu.

RET NZ :Jika hasil operasi ALU sebelumnya tidak sama dengan nol atau Z = 0 maka PC diisi nilai dari stack, kembali ke program utama. Untuk keadaan lain PC kontinyu

RET Z : Jika hasil operasi ALU sebelumnya sama dengan nol atau Z = 1 maka PC diisi nilai dari stack, kembali ke program utama. Untuk keadaan lain PC kontinyu

RET P : Jika tidak ada Sign flag atau S = 0 maka PC diisi nilai dari stack, kembali ke program utama. Untuk keadaan lain PC kontinyu

RET PE :Jika hasil operasi ALU sebelumnya paritasnya genap atau nilai flag P = 1 maka PC diisi nilai dari stack, kembali ke program utama. Untuk keadaan lain PC kontinyu.

RET PO : Jika hasil operasi ALU sebelumnya paritasnya ganjil atau nilai flag P = 0 maka maka PC diisi nilai dari stack, kembali ke program utama. Untuk keadaan lain PC kontinyu.

Disamping itu ada dua perintah RETURN lainnya yaitu :

RETI : Return from Interrupt

RETN1 : Return from Non Maskable Interrupt

10. Instruksi Restart

Instruksi Restart sama dengan instruksi CALL tanpa syarat, hanya instruksi Restart menunjuk ke alamat page zerro (halaman 00).

Instruksi **RST** menjalan proses :

(SP-1) ← PCH
(SP-2) ← PCL
PCH ← 00h
PCL ← p

Dimana p adalah nilai 8 bit penunjuk alamat pada page zerro. Pada instruksi Restart alamat yang dipanggil sudah tertentu yaitu :

RST 00h memanggil alamat 0000h
RST 08h memanggil alamat 0008h
RST 10h memanggil alamat 0010h
RST 18h memanggil alamat 0018h
RST 20h memanggil alamat 0020h
RST 28h memanggil alamat 0028h
RST 30h memanggil alamat 0030h
RST 38h memanggil alamat 0038h

9. Instruksi Input Output

Instruksi INPUT dan OUTPUT sangat duperlukan pada pembentukan program interface ke unit I/O. Instruksi input digunakan untuk mengambil data dari Port Input, sedangkan instruksi Output digunakan untuk mengirim data ke Port Output. Pengalamatan I/O dapat menggunakan pengalamat langsung (direct) atau tidak langsung (indirect). Ada 12 bentuk instruksi Input-Output seperti Gambar 6.49 berikut.

Ops	Assembly	Simbol Operasi	Ket
8 Bit	IN A, (N)	A ← (N)	Mempengaruhi Flag S, Z, H,
	IN A, (C)	A ← (C)	
	IN B, (C)	B ← (C)	
	IN C, (C)	C ← (C)	
	IN D, (C)	D ← (C)	
	IN E, (C)	E ← (C)	
	IN H, (C)	H ← (C)	
	IN L, (C)	L ← (C)	
	INI	(HL) ← (C)	
		B ← B-1	
		HL ← HL+1	
	INIR	(HL) ← (C)	
		B ← B-1	
		HL ← HL+1	
		Repeat until B = 0	
		(HL) ← (C)	

Ops	Assembly	Simbol Operasi	Ket
	IND	B ← B-1 HL ← HL-1 (HL) ← (C)	
	INDR	B ← B-1 HL ← HL-1 Repeat until B = 0	
	OUT (N), A	(N) ← A	
	OUT (C), A	(C) ← A	
	OUT (C), B	(C) ← B	
	OUT (C), C	(C) ← C	
	OUT (C), D	(C) ← D	
	OUT (C), E	(C) ← E	
	OUT (C), H	(C) ← H	
	OUT (C), L	(C) ← L	
	OUTI	(C) ← (HL) B ← B-1 HL ← HL+1	
	OTIR	(C) ← (HL) B ← B-1 HL ← HL+1 Repeat until B = 0	
	OUTD	(C) ← (HL) B ← B-1 HL ← HL-1	
	OTDR	(C) ← (HL) B ← B-1 HL ← HL-1 Repeat until B = 0	

Gambar 6.49. Contoh instruksi Input-Output

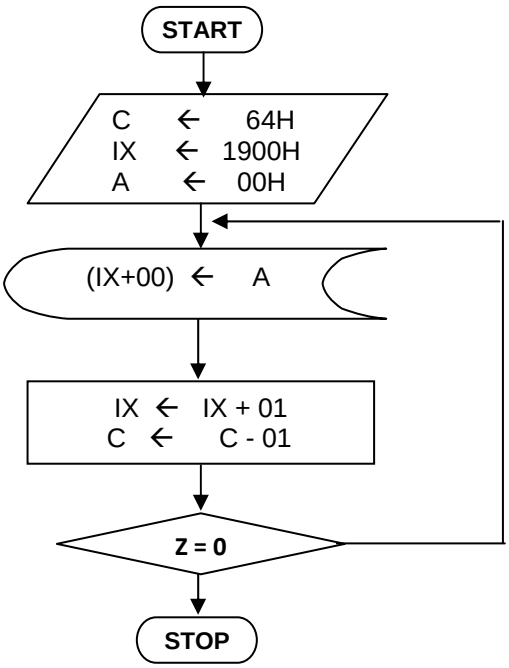
Berikut disajikan beberapa kasus program sebagai bahan kajian penerapan instruksi-instruksi yang sudah dijelaskan sebelumnya.

Kasus.

Algoritma Program, Flow Chart dan Program untuk mengosongkan 100 lokasi memori mulai alamat 1900.

Algoritma :

1. *Buat cacahan jumlah data sama dengan 100 (64h) di Register C*
2. *Set alamat awal tempat simpan data di 1900h di Register IX*
3. *Set data awal sama dengan 0 di Register A.*
4. *Simpan data di Reg. A ke memori tempat simpan data Reg IX*
5. *Naikkan nilai pencatat alamat Reg. IX satu alamat*
6. *Kurangi data cacahan Reg C dengan 1*
7. *Lihat apakah nilai cacahan Reg C tidak sama dengan 0 (Zerro=0), jika ya kembali ke langkah 4.*
8. *Berhenti*



Gambar 6.50. Flowchart program mengosongkan 100 lokasi memori

Program :

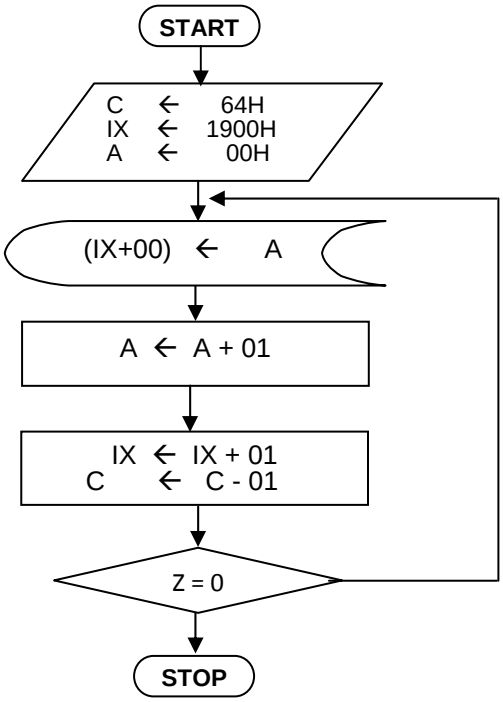
ADDR	Kode Operasi	No.	Label	Assembly	Ket.
1800	0E 64	1		LD C , 64H	
1802	DD 21 00 19	2		LD IX , 1900	
1806	3E 00	3		LD A , 00	
1808	DD 77 00	4	Catat	LD (IX+00), A	
180B	DD 23	5		INC IX	
180D	0D	6		DEC C	
180E	C2 08 18	7		JP NZ, Catat	
1811	FF	8		RST 38	

Gambar 6.51. Program mengosongkan 100 lokasi memori mulai alamat 1900

Algoritma Program, Flow Chart dan Program untuk membangkitkan 100 data bilangan desimal mulai dari 0 di memori mulai alamat 1900.

Algoritma :

1. *Buat cacahan jumlah data sama dengan 100 (64h) di Register C*
2. *Set alamat awal tempat simpan data di 1900h di Register IX*
3. *Set data awal sama dengan 0 di Register A.*
4. *Simpan data di Reg. A ke memori tempat simpan data Reg IX*
5. *Jumlahkan data yang ada pada Reg. A dengan 1*
6. *Desimalkan*
7. *Naikkan nilai pencatat alamat Reg. IX satu alamat*
8. *Kurangi data cacahan Reg C dengan 1*
9. *Lihat apakah nilai cacahan Reg C tidak sama dengan 0, jika ya kembali ke langkah 4.*
10. *Berhenti*



Gambar 6.52. Flowchart program membangkitkan 100 data bilangan desimal mulai alamat 1900

Program :

ADDR	Kode Operasi	No.	Label	Assembly	Ket
1800	0E 64	1		LD C, 64H	
1802	DD 21 00 19	2		LD IX, 1900H	
1806	3E 00	3		LD A , 00H	
1808	DD 77 00	4	Catat	LD IX+00) , A	
180B	C6 01	5		ADD A, 01	
180D	27	6		DAA	
180E	DD 23	7		INC IX	
1810	0D	8		DEC C	
1811	C2 08 18	9		JP NZ, Catat	
1814	FF	10		RST 38	

Gambar 6.53 Program membangkitkan 100 data desimal lokasi memori mulai alamat 1900

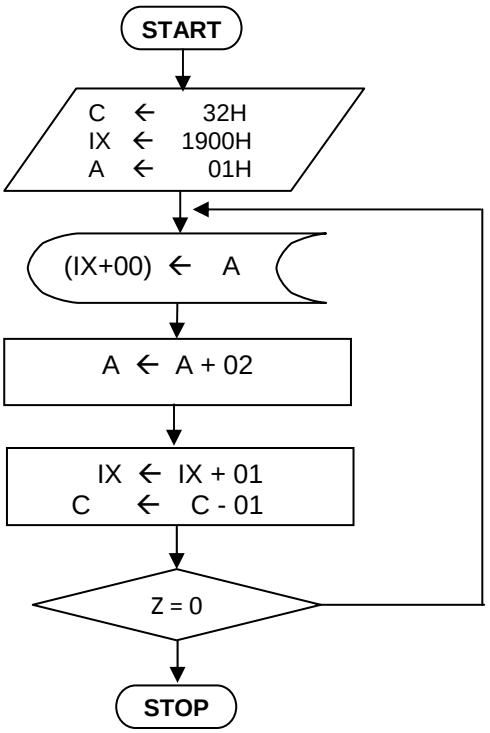
Algoritma Program, Flow Chart dan Program untuk membangkitkan 50 data bilangan desimal ganjil mulai dari 1 di memori mulai alamat 1900.

Algoritma :

1. *Buat cacahan jumlah data sama dengan 50 (32h) di Register C*
2. *Set alamat awal tempat simpan data di 1900h di Register IX*
3. *Set data awal sama dengan 1 di Register A.*
4. *Simpan data di Reg. A ke memori tempat simpan data Reg IX*
5. *Jumlahkan data yang ada pada Reg. A dengan 2*
6. *Naikkan nilai pencatat alamat Reg. IX satu alamat*
7. *Kurangi data cacahan Reg C dengan 1*
8. *Lihat apakah nilai cacahan Reg C tidak sama dengan 0, jika ya kembali ke langkah 4.*
9. *Berhenti*

Untuk membentuk hitungan cacahan pengulangan sebanyak 50 kali diperlukan perhitungan konversi bilangan desimal 50 menjadi 32 heksa desimal. Alamat awal tempat simpan data ditetapkan terlebih dahulu pada daerah RWM yakni alamat 1900h. Selanjutnya dengan perintah increament alamat memori tempat simpan data bertambah satu dan berulang sebanyak 50 kali melakukan proses penambahan data awal A= 1 dengan 2. Cara ini sangat memudahkan dan membuat proses menjadi sederhana.

Flow Chart



Gambar 6.54. Flowchart membangkitkan 50 bilangan desimal ganjil mulai alamat 1900

Program :

ADDR	Kode Operasi	No	Label	Assembly	Ket
1800	0E 32	1		LD C , 32H	
1802	DD 21 00 19	2		LD IX , 1900	
1806	3E 01	3		LD A , 01	
1808	DD 77 00	4	Catat	LD (IX+00) , A	
180B	C6 02	5		ADD A , 02	
180D	27	6		DAA	
180E	DD 23	7		INC IX	
1810	0D	8		DEC C	
1811	C2	9		JP NZ, Catat	
1812	FF	10		RST 38	

Gambar 6.55 Program membangkitkan 50 bilangan desimal ganjil mulai alamat 1900

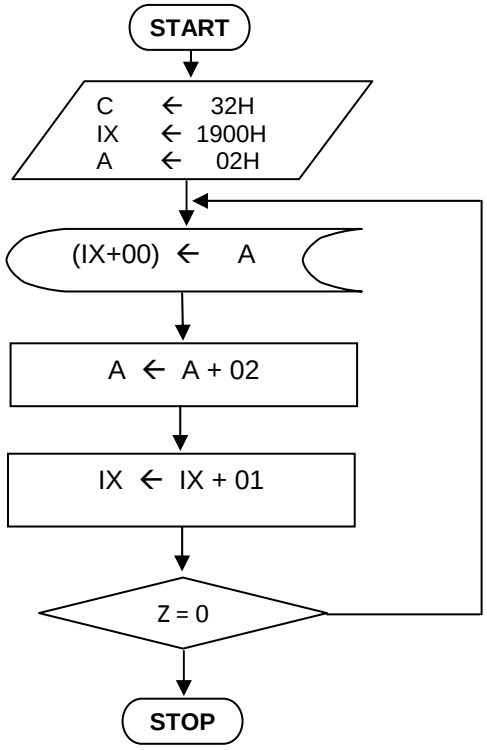
Algoritma Program, Flow Chart dan Program untuk membangkitkan 50 data bilangan desimal genap mulai dari 2 di memori mulai alamat 1900.

Algoritma :

1. *Buat cacahan jumlah data sama dengan 50 (32h) di Register C*
2. *Set alamat awal tempat simpan data di 1900H di Register IX*
3. *Set data awal sama dengan 2 di Register A.*
4. *Simpan data di Reg. A ke memori tempat simpan data Reg IX*
5. *Jumlahkan data yang ada pada Reg. A dengan 2*
6. *Naikkan nilai pencatat alamat Reg. IX satu alamat*
7. *Kurangi data cacahan Reg C dengan 1*
8. *Lihat apakah nilai cacahan Reg C tidak sama dengan 0, jika ya kembali ke langkah 4.*
9. *Berhenti*

Untuk membentuk hitungan cacahan pengulangan sebanyak 50 kali diperlukan perhitungan konversi bilangan desimal 50 menjadi 32 heksa desimal. Alamat awal tempat simpan data ditetapkan terlebih dahulu pada daerah RWM. Selanjutnya dengan perintah increament alamat memori tempat simpan data bertambah satu dan berulang sebanyak 50 kali. Pembangkitan data desimal genap dilakukan dengan menambahkan nilai awal A= 2 dengan 2. Cara ini sangat memudahkan dan membuat proses menjadi sederhana.

Flow Chart



Gambar 6.56. Flowchart membangkitkan 50 bilangan desimal genap mulai alamat 1900

Program :

ADD	Kode Operasi	No.	Label	Mnemonik	Ket
1800	0E 32	1		LD C, 32H	
1802	DD 21 00 19	2		LD IX, 1900	
1806	3E 02	3		LD A, 02	
1808	DD 77 00	4	Catat	LD (IX+00), A	
180B	C6 02	5		ADD A, 02	
180D	27	6		DAA	
180E	DD 23	7		INC IX	
1810	0D	8		DEC C	
1811	C2	9		JP NZ, Catat	
1812	FF	10		RST 38	

Gambar 6.57. Flowchart membangkitkan 50 bilangan desimal genap mulai alamat 1900