INSTRUKSI-INSTRUKSI MIKROPROSESOR Z80

Yoyo somantri
Dosen Jurusan Pendidikan Teknik Elektro
FPTK Universitas Pendidikan Indonesia

Pendahuluan

Pada bab ini akan dibahas tujuan perkuliahan, instruksi yang terdapat pada mikroprosesor Z80 yang terdiri dari kelompok *Load* 8 bit, *Load* 16 bit, *Exchange*, *block transfer* dan *search*, Operasi – operasi *arithmetic* dan *logic* 8 bit serta 16 bit, *Rotate* dan *shift*, Bit *set*, *reset*, dan operasi – operasi *test*, *Jump*, *Call*, *return* dan *restart*, dan Operasi – operasi *input* dan *output*.

Tujuan Perkuliahan

Setelah mempelajari bab ini, diharapkan mahasiswa mampu untuk:

1. Memahami instruksi- instruksi yang terdapat pada mikroprosesor Z80 yang diawali dari Load, exchange, block transfer, search, arithmetic, logika, rotasi, shift, jump, call, return, dan opearsi-input-output.
2. Mengaplikasikan set instruksi dalam menyelesaikan suatu masalah dalam pembuatan program.

1. Instruksi Z 80

Mikroprosesor Z80 mempunyai 158 jenis instruksi. Dengan adanya berbagai mode pengalamatan pada mikroprosesor Z80, maka dapat diperoleh total sebanyak 694 instruksi yang berbeda yang dapat dipakai dalam pemograman mikroprosesor Z80.

Instruksi-instruksi yang digunakan pada mikroprosesor Z80 dapat dikelompokkan atas 11 kelompok, yaitu :

- *Load* 8 bit
- *Load* 16 bit
- *Exchange, block transfer* dan *search*
- Operasi – operasi *arithmetic* dan *logic* 8 bit
- *Arithmetic* serbaguna dan kontrol CPU
- Operasi - operasi *arithmetic* 16 bit
- *Rotate* dan *shift*
- *Bit set*, *reset*, dan operasi – operasi *test*.
- *Jump*
- *Call, return* dan *restart*
- Operasi – operasi input dan output.

Tabel 1 . Kelompok Load 8 Bit

Sumber dari Buku Guru Mikro saya (1986) Inelco dan buku Laventhal, (1986). Z80 Assembly Language Programming.

Mnemonic	**Operasi**	**Keterangan**
LD r, r'	r, ← r'	Isi *Register* r' diisikan ke *register* r
LD r,n	r ← n	Data 8 bit diisikan ke *Register* r
LD r, (HL)	r ← (HL)	Isi lokasi memori (HL) diisikan ke *Register* r
LD r, (IX+d)	r ← (IX+d)	Isi *Register Index* IX ditambah dengan bilangan bulat penggeser sebesar d diisikan ke *register* r
LD r, (IY+d)	r ← (IY+d)	Isi *Register Index* IY ditambah dengan bilangan bulat penggeser sebesar d diisikan ke register r
LD (HL), r	(HL) ← r	Isi *Register* r diisikan ke lokasi memori yang ditunjuk oleh isi pasangan *Register* HL
LD (IX+d),	(IX+d) ← r	Isi *Register* r diisikan ke lokasi memori yang

r		ditunjuk oleh *Register Index* IX + d
LD (IY+d), r	(IY+d) ← r	Isi *Register* r diisikan ke lokasi memori yang ditunjuk oleh *Register Index* IY + d
LD (HL), n	(HL) ← n	Isi data n diisikan ke alamat memori yang ditunjuk oleh pasangan *Register* HL
LD (IX+d), n	(IX+d) ← n	Isi data n diisikan ke alamat memori yang ditunjuk oleh *Register Index* IX + d
LD (IY+d), n	(IY+d) ← n	Isi data n diisikan ke alamat memori yang ditunjuk oleh *Register Index* IY + d
LD A, (BC)	A ← (BC)	Isi lokasi memori yang ditunjuk oleh isi pasangan *Register* BC diisikan ke Akumulator
LD A, (DE)	A ← (DE)	Isi lokasi memori yang ditunjuk oleh isi pasangan *Register* DE diisikan ke Akumulator
LD A, (nn)	A ← (nn)	Isi lokasi memori yang ditunjuk oleh isi *operand* nn diisikan ke Akumulator
LD (BC), A	(BC) ← A	Isi Akumulator diisikan ke lokasi memori yang ditunjuk oleh isi pasangan *Register* BC
LD (DE), A	(DE) ← A	Isi Akumulator diisikan ke lokasi memori yang ditunjuk oleh isi pasangan *Register* DE
LD (nn), A	(nn) ← A	Isi Akumulator diisikan ke lokasi memori yang ditunjuk oleh *operand* nn
LD A, I	A ← I	Isi *Register Interupt Vector* diisikan ke Akumulator
LD A, R	A, ← R	Isi *Register Memory Refresh* diisikan ke Akumulator
LD I,A	I ← A	Isi Akumulator diisikan ke *Register Interupt Vector*
LD R, A	R ← A	Isi Akumulator diisikan ke *Register Register Memory Refresh*

r, r’ *adalah register –register* A, B, C, D, H, L,D,E,F

Instruksi *load* 8 bit mempunyai *opcode* LD dan selalu mempunyai dua buah *operand*. Instruksi ini menyebabkan nilai yang dispesifikasikan oleh *operand* kedua disalin ke dalam *register* atau tempat tujuan lain yang dispesifikasikan oleh *operand* pertama, sedangkan *operand* kedua tidak berubah.

Contoh :

1. Jika *register* H berisi bilangan BAH, dan register E berisi 11H, instruksi LD H, E, mengakibatkan kedua *register* berisi 11H.
2. Tulislah program dalam bahasa *assembly* untuk mengeset isi register A=A, B=1, C=4, D=3, E=4, H=8, L=6.

Jawab :

Alamat memori	Bahasa *Assembly*
1800H	LD A,A
1802H	LD B,1
1804H	LD C,4
1806H	LD D,3
1808H	LD E,4
180AH	LD H,8
180CH	LD L,6
180EH	RST 38

Register 8 bit akan disebut r, sedangkan *register* 16 bit akan disebut rr. Bilangan 8 bit akan disebut n, sedangkan bilangan 16 bit akan disebut nn. Isi suatu lokasi memori dinyatakan dengan tanda kurung, contoh :

(nn) : Menyatakan isi lokasi memori yang alamatnya nn.

(rr) : Menyatakan isi lokasi memori yang ditunjukkan oleh isi *register* 16 bit (rr) atau register pasangan.

Tabel 2. Kelompok *Load* 16 bit

Sumber dari Buku Guru Mikro saya (1986) Inelco dan buku Laventhal, (1986). Z80 Assembly Language Programming.

Mnemonic	**Operasi**	**Keterangan**
LD dd, nn	dd ← nn	Bilangan bulat 2 byte diisikan ke pasangan *Register* dd
LD IX, nn	Ix ← nn	Bilangan bulat 2 byte diisikan ke *Register Index* IX.
LD IY, nn	IY ← nn	Bilangan bulat 2 byte diisikan ke *Register Index* IY.
LD HL, (nn)	H ← (nn+1), L ← (nn)	Isi *address memory* nn diisikan pada byte yang berorde rendah dari pasangan *Register* HL(dalam hal ini ke *Register* L)
LD dd, (nn)	ddh ← (nn+1) ddl ← (nn)	Isi *address memory* nn diisikan pada byte yang berorde rendah dari pasangan *Register* dd, dan isi dari *address memory* yang lebih tinggi nn+1 diisikan pada byte dd yang berorde tinggi.
LD IX, (nn)	IXh ← (nn+1), IXI ← (nn)	Isi *address memory* nn diisikan pada byte yang berorde rendah dari *Register Index* IX, dan isi dari *address memory* yang lebih tinggi nn+1 diisikan pada byte IX yang

		berorde tinggi.
LD IY, (nn)	IYh ← (nn+1), IYI ← nn)	Isi *address memory* nn diisikan pada byte yang berorde rendah dari Register Index IY, dan isi dari *address memory* yang lebih tinggi nn+1 diisikan pada byte IY yang berorde tinggi.
LD (nn), HL	(nn+1) ← H, (nn) ← L	Isi pasangan *Register* HL yang berorde rendah (*Register* L) diisikan ke *address memory* nn, dan isi pasangan *Register* HL yang berorde tinggi (*Register* H) diisikan pada *address memory* nn+1 berikutnya yang tinggi.
LD (nn), dd	(nn+1) ← ddh, (nn) ← ddl	Isi pasangan *Register* dd yang berorde rendah diisikan ke address memory nn, dan isi pasangan *Register* dd yang berorde tinggi diisikan pada address memory nn+1 berikutnya yang tinggi.
LD (nn), IX	(nn+1) ← IXh, (nn) ← IXI	Byte berorde rendah dari *Register Index* IX diisikan ke *address memory* nn, dan isi *Register index* IX yang berorde tinggi diisikan pada *address memory* nn+1
LD (nn), IY	(nn+1) ← IYh, (nn) ← IYI	Byte berorde rendah dari *Register Index* IY diisikan ke *address memory* nn, dan isi *Register index* IY yang berorde tinggi diisikan pada *address memory* nn+1
LD SP, HL	SP ← HL	Isi pasangan *Register* HL diisikan ke *Stack Ponter* SP
LD SP, IX	SP ← IX	Isi *Register Index* IX yang terdiri dari 2 byte tersebut diisikan ke *Stack*

		Pointer SP
LD SP, IY	SP ← IY	Isi *Register Index* IY yang terdiri dari 2 byte tersebut diisikan ke *Stack Pointer* SP
PUSH qq	(SP-2) ← qq_R, (SP -1) ←qq_T	Isi pasangan *register* qq disimpan dalam '*eksternal memory* LIFO (*last in first out*/ terakhir masuk pertama keluar) *stack'. Register* SP (*stack pointer*) menyimpan address 16 bit dari bagian atas (terakhir)*stack memory*. Instruksi ini mula-mula menurunkan SP (mengurangi SP dengan 1) dan menyimpan byte berorde tinggi dari pasangan register qq ke memori yang sekarang addressnya ditunjuk oleh oleh SP. Lalu menurunkan SP lagi dan menyimpan byte qq yang berorde rendah ke memory yang *address*-nya ditunjukkan oleh nilai SP yang baru. *Operand* qq = pasangan *register* BC, DE, HL atau AF.
PUSH IX	(SP-2) ← IX_R, (SP -1) ←IX_T	Isi *register indeks* IX disimpan dalam 'eksternal memory LIFO (*last in first out*/ terakhir masuk pertama keluar) *stack'. Register* SP (*stack pointer*) menyimpan *address* 16 bit dari bagian atas (terakhir)*stack memory*. Instruksi ini mula-mula menurunkan SP (mengurangi SP dengan 1) dan menyimpan byte berorde tinggi dari *register* indeks IX ke memori yang sekarang addressnya ditunjuk oleh oleh SP. Lalu menurunkan SP lagi dan menyimpan byte IX yang

		berorde rendah ke memori yang *address*-nya ditunjukkan oleh nilai SP yang baru.
PUSH IY	(SP-2) ← IY_R, (SP -1) ←IY_T	Isi *register* indeks IY disimpan dalam ‘eksternal memory LIFO (*last in first out*/ terakhir masuk pertama keluar) stack’. Register SP (*stack pointer*) menyimpan *address* 16 bit dari bagian atas (terakhir)*stack memory*. Instruksi ini mula-mula menurunkan SP (mengurangi SP dengan 1) dan menyimpan byte berorde tinggi dari *register* indeks IY ke memory yang sekarang *address*-nya ditunjuk oleh oleh SP. Lalu menurunkan SP lagi dan menyimpan byte IY yang berorde rendah ke memori yang addressnya ditunjukkan oleh nilai SP yang baru.
POP qq	qq_T← (SP +1), qq_R ← (SP)	Dua byte teratas dari ‘*eksternal memory* LIFO (*last in first out*/ terakhir masuk pertama keluar) *stack*’. Dipindahkan ke *register* qq. *Register* SP (*stack pointer*) menyimpan *address* 16 bit dari bagian atas (terakhir)*stack memory*. Instruksi ini mula-mula memindahkan byte yang ada pada memory yang yang *address*-nya ditunjuk oleh SP ke byte pada qq yang berorde rendah. Lalu SP dinaikkan(nilai SP ditambah dengan 1). Byte qq yang berorde tinggi lalu diisi dengan byte pada memory yang addressnya ditunjukkan oleh nilai SP yang baru.

POP IX	$IX_T \leftarrow$ (SP +1), $IX_R \leftarrow$ (SP)	Dua byte teratas dari ‘eksternal *memory* LIFO (*last in first out*/ terakhir masuk pertama keluar) *stack’*. Dipindahkan ke *register* indeks IX. *Register* SP (*stack pointer*) menyimpan *address* 16 bit dari bagian atas (terakhir)*stack memory*. Instruksi ini mula-mula memindahkan byte yang ada pada memory yang yang *address*-nya ditunjuk oleh SP ke byte pada IX yang berorde rendah. Lalu SP dinaikkan(nilai SP ditambah dengan 1). Byte IX yang berorde tinggi lalu diisi dengan byte pada memori yang *address*-nya ditunjukkan oleh nilai SP yang baru. Nilai SP ditambah lagi dengan 1
POP IY	$IY_T \leftarrow$ (SP +1), $IY_R \leftarrow$ (SP)	Dua byte teratas dari ‘eksternal *memory* LIFO (*last in first out*/ terakhir masuk pertama keluar) *stack*’. Dipindahkan ke *register* indeks IY. *Register* SP (*stack pointer*) menyimpan *address* 16 bit dari bagian atas (terakhir)*stack memory*. Instruksi ini mula-mula memindahkan byte yang ada pada memori yang *address*-nya ditunjuk oleh SP ke byte pada IY yang berorde rendah. Lalu SP dinaikkan(nilai SP ditambah dengan 1). Byte IY yang berorde tinggi lalu diisi dengan byte pada memory yang *address*-nya ditunjukkan oleh nilai SP yang baru. Nilai SP ditambah lagi dengan 1

Seperti halnya instruksi *load* 8 bit, pada instruksi *load* 16 bit juga mempunyai *op code* LD dan dua buah *operand*. Instruksi ini menyebabkan nilai 16 bit yang dispesifikasikan oleh *operand* kedua disalin ke tempat tujuan 16 bit yang dispesifikasikan oleh operand pertamanya. *Flag* tidak dipengaruhi oleh instruksi 16 bit. Contoh :

1. Setelah pelaksanaan instruksi LD HL, 5000H, isi pasangan HL = 5000H.

2. Jika alamat 2130H berisi 65H dan alamat 2131H berisi 78H, setelah pelaksanaan instruksi LD BC, (2130H), isi pasangan *register* BC akan berisi 7865H.

3. Jika pasangan *register* AF berisi 2233H dan *Stack Pointer* berisi 1007H, setelah instruksi PUSH AF, maka alamat memori 1006H akan berisi 22H, alamat memori 1005H akan berisi 33H, dan *Stack Pointer* akan berisi 1005H.

4. Jika *Stack Pointer* berisi 1000H, memori pada lokasi 1000H berisi 55H dan lokasi 1001H berisi 35H, maka setelah instruksi POP HL, pasangan *register* HL berisi 3355H dan *Stack Pointer* berisi 1002H.

5. Tulislah program dalam bahasa *assembly* untuk mengeset isi register B=12, C=34, D=56, E=78, H=9, L=A.

Jawab :

Alamat memori	Bahasa *Assembly*
1820H	LD BC,1234H
1823H	LD DE,5678H
1826H	LD HL,9A
1828H	RST 38

Tabel 3. Kelompok *Exchange, Block Transfer* dan *Search*

Sumber dari Buku Guru Mikro saya (1986) Inelco dan buku Laventhal, (1986). Z80 Assembly Language Programming.

Mnemonic	**Operasi**	**Keterangan**
EX DE, HL	DE ↔ HL	Isi pasangan *register* DE dan HL (masing-masing 2 byte) dipertukarkan
EX AF, AF'	AF ↔ AF'	Isi pasangan *register* AF dan AF' (masing-masing 2 byte) dipertukarkan
EXX	(BC) ↔ (BC'), (DE) ↔ (DE'), (HL) ↔ (HL')	Isi pasangan *register* BC dan BC' (masing-masing 2 byte) dipertukarkan, dst.
EX (SP), HL	H ↔ (SP+1), L ↔ (SP)	Byte berorde rendah yang berada pada pasangan register HL dipertukarkan dengan isi memori yang *address*-nya ditunjuk oleh isi pasangan *register* SP, byte yang berorde tinggi pada HL dipertukarkan dengan isi memori yang *address*-nya ditunjuk oleh SP+1
EX (SP), IX	IXH ↔ (SP+1), IXL ↔ (SP)	Byte berorde rendah yang berada pada *register index* IX dipertukarkan dengan isi memori yang *address*-nya ditunjuk oleh isi pasangan register SP, byte yang berorde tinggi pada IX dipertukarkan dengan isi memori yang *address*-nya ditunjuk oleh SP+1
EX (SP), IY	IYH ↔ (SP+1), IYL ↔ (SP)	Byte berorde rendah yang berada pada *register index* IY dipertukarkan dengan isi memori yang *address*-nya ditunjuk oleh isi pasangan *register* SP, byte yang berorde tinggi pada IY dipertukarkan dengan isi memori yang *address*-nya ditunjuk oleh

		SP+1
LDI	(DE) ← (HL), DE ← DE + 1, HL ← HL + 1, BC ← BC -1	Satu byte data pada memori yang lokasinya ditunjuk oleh isi pasangan register HL diisikan ke lokasi memori yang lokasinya ditunjukkan oleh isi pasangan register DE. Lalu kedua pasangan *register* tersebut(HL dan (DE) ditambah dengan satu (*incremented*) dan pasangan *register* BC (*Byte Counter*) dikurangi dengan satu (*decremented*)
LDIR	(DE) ← (HL), DE ← DE + 1, HL ← HL + 1, BC F↔ BC -1	Satu byte data pada memori yang lokasinya ditunjuk oleh isi pasangan *register* HL diisikan ke lokasi memori yang lokasinya ditunjukkan oleh isi pasangan *register* DE. Lalu kedua pasangan *register* tersebut (HL dan (DE) ditambah dengan satu (*incremented*) dan pasangan *register* BC (*Byte Counter*) dikurangi dengan satu (*decremented*). Jika penurunan tersebut menyebabkan BC menjadi 0, instruksi ini dianggap selesai (dihentikan). Jika BC belum menjadi 0, program counter diturunkan dengan 2 dan instruksi tersebut diulangi lagi. Perhatikan bahwa bilamana BC di set 0 sebelum pelaksanaan instruksi ini, instruksi dilaksanakan sebanyak 64K kali. Interupsi akan dilayani setiap selesai mentransfer 1 byte data.
LDD	(DE) ← (HL), DE ← DE -1, HL ← HL-1, BC ← BC-1	Satu byte data pada memori yang lokasinya ditunjuk oleh isi pasangan register HL diisikan ke lokasi memori yang lokasinya ditunjukkan oleh isi pasangan register DE. Lalu kedua pasangan *register* tersebut (HL dan (DE), juga BC dikurangi dengan 1 (*decremented*)

LDDR	(DE) ← (HL), DE ← D ← 1, HL ← HL-1, BC ← BC-1	Satu byte data pada memory yang lokasinya ditunjuk oleh isi pasangan *register* HL diisikan ke lokasi memori yang lokasinya ditunjukkan oleh isi pasangan *register* DE. Lalu kedua pasangan *register* tersebut (HL dan (DE), juga BC dikurangi dengan 1 (*decremented*) Jika penurunan tersebut menyebabkan BC menjadi 0, instruksi ini dianggap selesai (dihentikan). Jika BC belum menjadi 0, program counter diturunkan dengan 2 dan instruksi tersebut diulangi lagi. Perhatikan bahwa bilamana BC di set 0 sebelum pelaksanaan instruksi ini, instruksi dilaksanakan sebanyak 64K kali. Interupsi akan dilayani setiap selesai mentransfer 1 byte data.
CPI	A- (HL), HL ← HL +1, BC ← BC -1	Isi memori yang lokasinya ditunjukkan oleh isi pasangan *register* HL dibandingkan dengan isi akumulator. Jika hasilnya sama, ada kondisi bit yang dipengaruhi. Kemudian isi HL akan ditambah dengan 1 dan byte counter (pasangan *register* BC) dikurangi dengan 1.
CPIR	A-(HL), HL ← HL+1, BC ← BC-1	Isi memori yang lokasinya ditunjukkan oleh isi pasangan *register* HL dibandingkan dengan isi akumulator. Jika hasilnya sama, ada kondisi bit yang dipengaruhi. Kemudian isi HL akan ditambah dengan 1 dan *byte counter* (pasangan register BC) dikurangi dengan 1. Jika penurunan tersebut menyebabkan BC menjadi 0 atau jika A =(HL), instruksi ini dianggap selesai (dihentikan). Jika BC belum menjadi 0, *program counter* diturunkan dengan 2 dan instruksi tersebut diulangi lagi. Perhatikan bahwa bilamana BC di set 0 sebelum pelaksanaan instruksi ini, instruksi

		dilaksanakan sebanyak 64K kali. Interupsi akan dilayani setiap selesai mentransfer 1 byte data. Jika tidak ada yang cocok , interupsi akan dilayani setiap satu data selesai dibandingkan.
CPD	A -(HL), HL ← HL -1, BC ← BC -1	Isi memori yang lokasinya ditunjukkan oleh isi pasangan *register* HL dibandingkan dengan isi akumulator. Jika hasilnya sama, ada kondisi bit yang dipengaruhi. Kemudian isi HL dan *byte counter* (pasangan *register* BC) dikurangi dengan 1. untuk BC tidak sama dengan 0 atau A tidak sama dengan HL
CPDR	A -(HL), HL ← HL -1, BC ← BC -1	Isi memori yang lokasinya ditunjukkan oleh isi pasangan *register* HL dibandingkan dengan isi akumulator. Jika hasilnya sama, ada kondisi bit yang dipengaruhi. Kemudian isi HL dan *byte counter* (pasangan *register* BC) dikurangi dengan 1. Jika penurunan tersebut menyebabkan BC menjadi 0 atau jika A =(HL), instruksi ini dianggap selesai (dihentikan). Jika BC belum menjadi 0, *program counter* diturunkan dengan 2 dan instruksi tersebut diulangi lagi. Perhatikan bahwa bilamana BC di set 0 sebelum pelaksanaan instruksi ini, instruksi dilaksanakan sebanyak 64K kali. Interupsi akan dilayani setiap selesai mentransfer 1 byte data. Jika tidak ada yang cocok , interupsi akan dilayani setiap satu data selesai dibandingkan.

Pada CPU Z 80, ada sekumpulan *register* cadangan, yaitu A'F', B'C', D'E', H'L', yang hanya dapat dicapai dengan suatu instruksi tukar (*exchange*). Suatu instruksi *exchange*, saling mempertukarkan nilai-nilai pada kedua *operand*. Contoh :

Jika isi pasangan BC dan DE berturut-turut adalah 445AH dan 3DA2H. Kemudian isi pasangan *register* BC' dan DE' adalah 0988H dan 9300H, maka setelah instruksi EXX, isi pasangan – pasangan *register* di atas akan menjadi : BC 0988H, DE 9300H, BC' 445AH dan DE' 3DA2H.

Tabel .4. Kelompok Arithmetik dan logik 8 bit.

Mnemonic	**Operasi**	**Keterangan**
ADD, A, r	A← A + 1	Isi *register* r ditambahkan pada isi akumulator, hasilnya disimpan pada akumulator. R = *register-register* A,B,C,D,E,H dan L.
ADD A, n	A← A + r	Bilangan bulat n ditambahkan pada isi akumulator, hasilnya disimpan pada akumulator.
ADD A, (HL)	A← A + (HL)	Byte pada memori yang lokasinya ditunjukkan oleh isi pasangan *register* HL ditambahkan pada isi akumulator, hasilnya disimpan pada akumulator.
ADD A, (IX + d)	A← A + (IX + d)	Isi *register* Indeks (pasangan *register* IX) yang ditambah dengan bilangan penggeser sebesar d akan menunjuk pada suatu lokasi pada memori. Isi pada lokasi inilah yang ditambahkan pada isi akumulator, hasilnya disimpan pada akumulator.

ADD A, (IY + d)	A← A + (IY + d)	Isi *register* Indeks (pasangan *register* IY) yang ditambah dengan bilangan penggeser sebesar d akan menunjuk pada suatu lokasi pada memori. Isi pada lokasi inilah yang ditambahkan pada isi akumulator, hasilnya disimpan pada akumulator.
ADC A, s	A← A + s + CY	Operand s adalah salah satu dari r, n, (HL), (IX + d), (IY + d) seperti yang telah diterangkan pada instruksi ADD. *Operand* s bersama *Flag Carry* (C pada register F) ditambahkan pada isi akumulator, hasilnya disimpan pada akumulator.
SUB s	A← A - s	Isi akumulator dikurangi dengan *operand* s, hasilnya disimpan pada akumulator
SBC s	A← A – s - CY	Isi akumulator dikurangi dengan *operand* s bersama dengan *Flag Carry* (C pada *register* F), hasilnya disimpan pada akumulator
AND s	A ← A ^s	Operasi logika AND. Bit demi bit dilakukan diantara byte yang ditunjuk oleh operand s dan byte yang ada pada akumulator, hasilnya disimpan pada akumulator.
OR s	A ← A $\vee$ s	Operasi logika OR. Bit demi bit dilakukan diantara byte yang ditunjuk oleh *operand* s dan byte yang ada pada akumulator, hasilnya disimpan pada akumulator
XOR a	$A \leftarrow A \oplus s$	Operasi logika EXOR. Bit demi bit dilakukan diantara byte yang ditunjuk oleh operand s dan byte yang ada pada akumulator, hasilnya disimpan pada akumulator.

CP s	A-s	Isi *operand* s dibandingkan dengan isi akumulator, jika cocok suatu *flag* di-set.
INC r	r ← r + 1	Isi register r ditambah dengan 1(incremented) , r = register A, B, C, D, H atau L.
INC (HL)	(HL) ← (HL)+ 1	Byte pada *address* (lokasi) yang ditunjuk oleh isi pasangan *register* HL ditambah dengan 1 (*incremented*).
INC (IX +d)	(IX+d) ← (IX+d) + 1	Isi *register* indeks IX ditambah dengan bilangan bulat penggeser sebesar d untuk menunjuk ke suatu lokasi pada memori. Isi pada lokasi inilah yang ditambah dengan 1 (*incremented*)
INC (IY +d)	(IY+d) ← (IY+d) + 1	Isi *register* indeks IY ditambah dengan bilangan bulat penggeser sebesar d untuk menunjuk ke suatu lokasi pada memori. Isi pada lokasi inilah yang ditambah dengan 1 (*incremented*)
DEC m	m ← m - 1	*Operand* m adalah salah satu dari r, (HL), (IX + d), (IY + d) seperti yang telah diterangkan pada instruksi ADD. Byte yang ditunjuk oleh operand m dikurangi dengan 1 (*decremented*)

Sumber dari Buku Guru Mikro saya (1986) Inelco dan buku Laventhal, (1986). Z80 Assembly Language Programming.

Contoh :

1. Tambahan isi program pada lokasi memori 1900 dan 19001 dan simpan hasilnya pada lokasi memori 1902

Contoh Masalah : (1900) = 38

(1901) = 2B

Hasil (1902) = ?

Jawab :

Program lengkap

Alamat memori (Hex)	Isi memori (Hex)	*Mnemonic* (substraksi) bahasa *assembly*
1800	3A	LD A, (1900)
1801	00	
1802	19	
1803	47	LD B, A
1804	3A	LD A, (1901)
1805	01	
1806	19	
1807	80	ADD A, B
1808	32	LD (1902), A
1809	02	
180A	19	
180B	76	Halt

Tabel 5. Arithmetik serbaguna dan control CPU

Sumber dari Buku Guru Mikro saya (1986) Inelco dan buku Laventhal, (1986). Z80 Assembly Language Programming.

Mnemonic	Operasi	Keterangan
DAA	---	Operasi ini akan menyesuaikan akumulator (bila syarat-syaratnya dipenuhi) untuk operasi penambahan dan pengurangan BCD.
CPL	A ← Â	Isi akumulator dibalikkan (*inverted*). Nol menjadi 1 dan 1 menjadi 0 (1 komplemen)
NEG	A ← 0-A	Isi akumulator dinegatifkan (2 komplemen) hal ini sama dengan mengurangi 0 dengan isi akumulator.
CCF	$CY \leftarrow \overline{CY}$	Flag C pada *register* F dibalikkan
SCF	CY ← 1	Flag C pada *register* F di-set
NOP	---	CPU tidak melakukan operasi apapun selama perioda mesin *(machine cycle*) ini.
HALT	---	Operasi HALT ini akan menunda operasi CPU sampai ada *interrupt* atau *reset* yang diterima. Selama dalam keadaan HALT prosesor akan melaksanakan NOP untuk memelihara kesegaran logik memori.
DI	IFF ← 0	DI tidak memungkinkan (*disable*) *maskable interrupt* dengan mereset *interrupt enable flip flop* (IFF1 dan IFF2). Perhatikan bahwa intruksi ini tidak memungkinkan *maskable interrupt* selama pelaksanaannya.
EI	IFF ← 1	EI memungkinkan (*enable*) *maskable interrupt* dengan mengeset *interrupt enable flip flop* (IFF1 dan IFF2). Perhatikan bahwa intruksi ini tidak memungkinkan maskable interrupt selama

		pelaksanaannya.
IM 0	---	Instruksi IM 0 ini mengeset *interrupt mode* 0. pada mode ini alat *interrupt* (*interrupt device*) dapat menyelipkan intruksi apa saja pada data bus dan memungkinkan CPU melaksanakannya.
IM 1	---	Instruksi IM 1 ini mengeset *interrupt mode* 1. pada mode ini prosesor akan melayani interrupt dengan melaksanakan start ulang pada lokasi (*address*) 0038H
IM 2	---	Instruksi IM 1 ini mengeset interrupt mode 2. mode ini memungkinkan panggilan tidak langsung ke lokasi mana saja pada memori. Dengan mode ini, CPU membentuk suatu ’16 bit memori address’. Delapan bit bagian atas adalah isi interrupt vector Register I dan delapan bir bagian bawah disuplay oleh alat *interrupt* ((*interrupt device*)

Contoh :

2. Komplemenkan isi pada lokasi memori 1900 dan simpan hasilnya ke dalam lokasi memori 1901

Contoh, Masalah (1900) = 6A

Hasil (1901) = ?

Program menggunakan bahas *assembly* Z.80

LD A, (1900) : kopikan data dari adres lokasi memori 1900 ke register A

CPL : Komplemenkan data pada register A

LD (1901), A : Simpan hasilnya pada lokasi memori 1901.

Halt : program berhenti

Jawab :

Program lengkap

Alamat memori (Hex)	Isi memori (Hex) *opcode*	Mnemonic (substraksi) bahasa *assembly*
1800	3A	LD A, (1900)
1801	00	
1802	19	
1803	2F	CPL
1804	32	LD (1901),A
1805	01	
1806	19	
1807	76	Halt

Instruksi DAA akan menyesuaikan akumulator (bila syarat-syaratnya dipenuhi) untuk operasi penambahan dan pengurangan BCD (*Binary Coded Decimal*). Untuk penambahan (ADD, ADC, INC) atau pengurangan (SUB, SBC, DEC, NEG).

Contoh :

Jika operasi penambahan dilaksanakan antara 15 (BCD) dan 27 (BCD), hasilnya dalam sistem desimal adalah 15 + 27 = 42. Tetapi angka-angka tadi diwakilkan dalam sistem biner dan dijumlahkan dalam akumlator menurut standar aritmetik ialah :

0011 1100

<u>+ 0000 0110</u>

0100 0010 = 42

Pada instruksi CPL, dimana isi register A dibalikkan (*inverted*), 1 menjadi 0, 0 menjadi 1 (1 komplemen). Instruksi NEG menyatakan isi register A dinegatifkan (2 komplemen)

Tabel .6. Operasi - operasi arithmetic 16 bit

Mnemonic	**Operasi**	**Keterangan**
ADD HL, ss	HL← HL + ss	Isi pasangan *register* ss (salah satu dari pasangan *register* BC, DE, HL atau SP) ditambahkan pada isi pasangan *register* HL dan hasilnya disimpan pada HL.
ADC HL, ss	HL← HL + ss + CY	Isi pasangan *register* ss (salah satu dari pasangan *register* BC, DE, HL atau SP) bersama dengan *flag carry* (*flag* C dalam *regsiter* F) ditambahkan pada isi pasangan *register* HL dan hasilnya disimpan pada HL.
SBC HL, ss	HL← HL - ss	Isi pasangan *register* HL dikurangi dengan isi pasangan *register* ss dikurang dengan *Flag Carry* hasilnya disimpan pada HL.
ADD IX, pp	IX← IX + pp	Isi pasangan *register* pp (salah satu dari isi pasangan *register* BC, DE, IX atau SP) ditambahkan pada isi *register* indeks IX, hasilnya disimpan pada IX.
ADD IY, rr	IY← IY + rr	Isi pasangan *register* rr (salah satu dari isi pasangan *register* BC, DE, IY atau SP) ditambahkan pada isi *register* indeks IY, hasilnya disimpan pada IY.
INC ss	ss ← ss + 1	Isi pasangan *register* ss (salah satu dari pasangan *register* BC, DE, HL atau SP) ditambahkan dengan 1.
INC IX	IX ← IX +	Isi *register* indeks IX ditambahkan dengan 1

	1	
INC Y	IY ← IY + 1	Isi *register* indeks IY ditambahkan dengan 1
DEC ss	ss ← ss - 1	Isi pasangan *register* ss (salah satu dari register BC, DE, HL atau SP) dikurangi dengan 1.
DEC IX	IX ← IX - 1	Isi *register* indeks IX dikurangi dengan 1
DEC IY	IY ← IY - 1	Isi *register* indeks IY dikurangi dengan 1

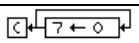
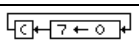
Contoh :

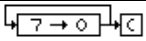
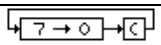
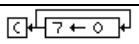
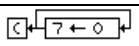
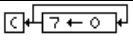
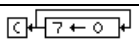
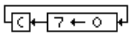
1. Jika pasangan *register* HL berisi bilangan bulat (*integer*) 4242H dan pasangan *register* DE berisi 1111H, setelah pelaksanaan ADD HL, DE, maka pasangan *register* HL berisi 5353H

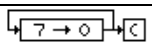
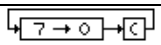
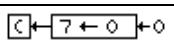
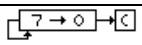
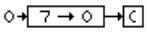
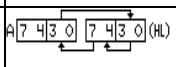
2. Jika pasangan *register* HL berisi 1000H, setelah pelaksanaan INC HL, maka HL akan berisi 1001H.

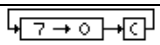
3. Jika *register* indeks IX berisi 2006H, setelah pelaksanaan DEC IX, maka IX akan berisi 2005H.

Tabel 7. *Rotate* dan *shift*

Mnemonic	**Operasi**	**Keterangan**
RLCA		Isi akumulator dirotasi ke kiri; isi bit 0 dipindahkan ke bit 1; isi bit 1 yang terdahulu dipindahkan ke bit 2 dan seterusnya. Isi bit 7 dicopy pada *flag carry* dan juga ke bit 0. (bit 0 adalah bit berorde paling rendah)
RLA		Isi akumulator dirotasi ke kiri; isi bit 0 dipindahkan ke bit 1; isi bit 1 yang terdahulu dipindahkan ke bit 2 dan seterusnya. Isi bit 7 dicopy pada *flag carry* dan isi *flag carry* yang

		terdahulu dicopy ke bit 0.
RRCA		Isi akumulator dirotasi ke kanan; isi bit 7 dipindah ke bit 6; isi bit 6 terdahulu dipindah ke bit 5; dan seterusnya. Isi bit 0 dicopy ke bit 7 dan juga ke *flag carry*.
RRA		Isi akumulator dirotasi ke kanan; isi bit 7 dipindah ke bit 6; isi bit 6 terdahulu dipindah ke bit 5; dan seterusnya. Isi bit 0 dicopy ke *flag carry* dan isi *flag carry* yang terdahulu dicopy ke bit 7.
RLC r		Isi *register* r (8 bit) dirotasi ke kiri; isi bit 0 dipindahkan ke 1; isi bit 1 terdahulu dipindahkan ke bit 2; dan seterusnya. Isi bit 7 dicopy ke *flag carry* dan juga ke bit 0.
RLC (HL)		Isi memori pada lokasi yang ditunjuk oleh pasangan *register* HL dirotasi ke kiri; isi bit 0 dipindah ke bit 1; isi bit 1 terdahulu dipindah ke bit 2; dan seterusnya. Isi bit 7 dicopy ke *flag carry* dan juga ke bit 0.
RLC (IX+d)		Isi memori pada lokasi yang ditunjuk oleh jumlah isi *register* indeks IX dan bilangan bulat penggeser dua komplemen d dirotasi ke kiri; isi bit 0 dipindah ke bit 1; isi bit 1 terdahulu dipindah ke bit 2; dan seterusnya. Isi bit 7 dicopy ke *flag carry* dan juga ke bit 0.
RLC (IY+d)		Isi memori pada lokasi yang ditunjuk oleh jumlah isi *register* indeks IY dan bilangan bulat penggeser dua komplemen d dirotasi ke kiri; isi bit 0 dipindah ke bit 1; isi bit 1 terdahulu dipindah ke bit 2; dan seterusnya. Isi bit 7 dicopy ke *flag carry* dan juga ke bit 0.
RL m		Operand m adalah salah satu dari r, (HL), (IX+d), (IY+d) seperti yang telah dijelaskan pada instruksi RLC. Isi operand m dirotasi ke kiri; isi

		bit 0 dipindah ke bit 1; isi bit 1 terdahulu dipindah ke bit 2; dan seterusnya. Isi bit 7 dicopy ke *flag* dan isi *flag carry* yang terdahulu dicopy ke bit 0.
RRC m		Isi dari operand m dirotasi ke kanan; isi bit 7 dipindahkan ke bit 6; isi bit 6 terdahulu dipindahkan ke bit 5; dan seterusnya. Isi bit 0 dicopy ke *flag carry* dan juga ke bit 7.
RR m		Isi dari operand m dirotasi ke kanan; isi bit 7 dipindahkan ke bit 6; isi bit 6 terdahulu dipindahkan ke bit 5; dan seterusnya. Isi bit 0 dicopy ke *flag carry* dan isi *flag carry* yang terdahulu dicopy ke bit 7.
SLA m		Pergeseran aritmetik ke kiri dilaksanakan pada isi operand m; bit 0 direset, isi bit 0 terdahulu dipindahkan ke bit 1; isi bit 1 terdahulu dipindahkan ke bit 2; dan seterusnya. Isi bit 7 dicopy ke *flag carry*.
SRA m		Pergeseran aritmetik ke kanan dilaksanakan pada isi operand m; isi bit 7 terdahulu dipindahkan ke bit 6; isi bit 6 terdahulu dipindahkan ke bit 6; dan seterusnya. Isi bit 0 dicopy ke *flag carry*. Isi flag 7 yang terdahulu tidak berubah.
SRL m		Pergeseran aritmetik ke kanan dilaksanakan pada isi operand m; isi bit 7 terdahulu dipindahkan ke bit 6; isi bit 6 terdahulu dipindahkan ke bit 6; dan seterusnya. Isi bit 0 dicopy ke flag carry. Isi flag 7 di reset
RLD		Isi 4 bit berorde rendah (bit 3, 2, 1, 0) dari memori yang lokasinya ditunjukkan oleh isi pasangan *register* HL dicopy ke 4 bit berorde tinggi (bit 7, 6, 5, 4) dari lokasi memori yang sama; isi ke 4 bit berorde tinggi terdahulu di

		copy ke 4 bit akumulator berorde rendah; dan isi 4 bit akumulator berorde rendah yang terdahulu dicopy ke 4 bit berorde rendah dari memori pada lokasi yang ditunjukkan oleh HL. Isi bit-bit akumulator berorde tinggi tidak terpengaruh.
RRD		Isi 4 bit berorde rendah (bit 3, 2, 1, 0) dari memori yang lokasinya ditunjukkan oleh isi pasangan *register* HL dicopy ke 4 bit berorde tinggi (bit 7, 6, 5, 4) pada memori yang lokasinya ditunjukkan oleh pasangan *register* HL; dan isi 4 bit berorde tinggi dari (HL) dicopy ke 4 bit (HL) yang rendah. Isi 4 bit akumulator berorde tinggi tidak terpengaruh. Catatan: (HL) berarti memori yang lokasinya ditunjukkan oleh isi pasangan *register* HL.

Contoh :

1. Jika isi akumulator adalah

7 6 5 4 3 2 1 0 (*format flag*)

1 0 0 0 1 0 0 0

Setelah pelaksanaan RLCA, isi akumulator dan *flag carry* akan menjadi

C 7 6 5 4 3 2 1 0 (*format flag*)

1 0 0 0 1 0 0 0 1

2. Jika isi akumulator dan *flag carry* adalah

C 7 6 5 4 3 2 1 0

1 0 1 1 1 0 1 1 0

Setelah pelaksanaan RLA, isi akumulator dan *flag carry* akan menjadi

C 7 6 5 4 3 2 1 0

0 1 1 1 0 1 1 0 1

Tabel 8. *Bit set*, *reset*, dan operasi – operasi test

Mnemonic	Operasi	Keterangan
BIT b, r	$z \leftarrow \overline{r_b}$	Setelah pelaksanaan instruksi ini, *flag* z pada *register* F akan berisi komplemen (lawan) dari bit yang dimaksud dalam *register* yang dimaksud.
BIT b, (HL)	$z \leftarrow \overline{(HL)_b}$	Setelah pelaksanaan instruksi ini, *flag* z pada *register* F akan berisi komplemen (lawan) dari bit yang dimaksud dalam pasangan *register* HL.
BIT b, (IX+d)	$z \leftarrow \overline{(IX+d)_b}$	Setelah pelaksanaan instruksi ini, *flag* z pada *register* F akan berisi komplemen (lawan) dari bit yang dimaksud dalam isi memori pada lokasi yang ditunjukkan oleh jumlah isi *register* indeks IX dan bilangan bulat penggeser 2 komplemen d.
BIT b, (IY+d)	$z \leftarrow \overline{(IY+d)_b}$	Setelah pelaksanaan instruksi ini, *flag* z pada register F akan berisi komplemen (lawan) dari bit yang dimaksud dalam isi memori pada lokasi yang ditunjukkan oleh jumlah isi *register* indeks IY dan bilangan bulat penggeser 2 komplemen d.
SET b, r	$R_b \leftarrow 1$	Bit b (bit yang mana saja dari bit 7 sampai 0) di *register* r (*register* B, C, D, E, H, L atau A) di-set.
SET b, (HL)	$(HL)_b \leftarrow 1$	Bit b (bit yang mana saja dari bit 7 sampai 0) pada lokasi yang memorinya ditunjukkan oleh pasangan *register* HL di-set.

SET b, (IX+d)	$(IX+d)_b$ ← 1	Bit b (bit yang mana saja dari bit 7 sampai 0) pada lokasi memori yang ditunjukkan oleh jumlah *register* indeks IX bilangan bulat penggeser 2 komplemen d, di – set.
SET b, (IY+d)	$(IY+d)_b$ ← 1	Bit b (bit yang mana saja dari bit 7 sampai 0) pada lokasi memori yang ditunjukkan oleh jumlah *register* indeks IY bilangan bulat penggeser 2 komplemen d, di – set.
RES b, m	S_b ← 0	Bit b (bit yang mana saja dari bit 7 sampai 0) dari isi operand m (salah satu dari r, (HL), (IX+d), (IY+d) seperti yang telah diterangkan pada instruksi SET. Bit b pada operand m direset.

Contoh :

1. Jika bit 2 pada *register* B berisi 0, setelah pelaksanaan BIT 2, B, maka

   Flag Z pada *register* F berisi 1, bit 2 dalam register B tetap 0. bit 0 adalah bit berorde paling rendah.

2. Jika pasangan *register* HL berisi 4444H, dan bit 4 pada lokasi memori 4444H berisi 1, setelah pelaksanaan BIT 2, B. maka

   Flag Z pada *register* F berisi 0, bit 4 pada memori di lokasi 4444H tetap berisi 1.

Tabel 9. Kelompok Jump

Mnemonic	**Operasi**	**Keterangan**
JP nn	PC ← nn	Operand nn diisikan ke pasangan *register* PC (*program counter*) dan

		menunjuk ke *address* instruksi program berikutnya yang akan dilaksanakan.
JP cc, nn	PC ← nn	Jika cc memenuhi syarat, instruksi mengisikan operand nn ke pasangan register PC, dan program berlanjut dengan instruksi yang dimulai pada *address* nn. Jika cc tidak memenuhi syarat, *program counter* ditambah dengan 1 seperti biasanya, dan program berlanjut dengan instruksi selanjutnya. Syarat cc diprogram sebagai salah satu dari delapan status yang berhubungan dengan kondisi bit pada *register flag* (*register* F).
JR e	PC ← PC + e	Instruksi ini memungkinkan percabagang (loncatan) tak bersyarat ke segmen lain dari suatu program. Nilai dari bilangan e ditambahkan ke *program counter* (PC) dan instruksi berikutnya diambil dari lokasi yang ditunjukkan oleh PC yang baru. Loncatan ini dihitung dari *address opcode* instruksi tersebut dan berada diantara -126 sampai +129 byte. *Assembler* secara otomatis menyesuaikan dengan cara mengurangi PC dengan 2.
JR C, e	Jika C= 1, PC ← PC + e	Instruksi ini memungkinkan percabagang (loncatan) tak bersyarat ke segmen lain dari suatu program tergantung dari hasil *test Flag Carry*. Jika *Flag* = 1, nilai dari bilangan e ditambahkan ke *program counter* (PC) dan instruksi berikutnya diambil dari lokasi yang ditunjukkan oleh PC yang

		baru. Loncatan ini dihitung dari *address opcode* instruksi tersebut dan berada diantara -126 sampai +129 byte. *Assembler* secara otomatis menyesuaikan dengan cara mengurangi PC dengan 2.
JR NC, e	Jika C= 0, PC ← PC + e	Instruksi ini memungkinkan percabagang (loncatan) tak bersyarat ke segmen lain dari suatu program tergantung dari hasil *test Flag Carry*. Jika *Flag* = 0, nilai dari bilangan e ditambahkan ke *program counter* (PC) dan instruksi berikutnya diambil dari lokasi yang ditunjukkan oleh PC yang baru. Loncatan ini dihitung dari *address opcode* instruksi tersebut dan berada diantara -126 sampai +129 byte. *Assembler* secara otomatis menyesuaikan dengan cara mengurangi PC dengan 2.
JR Z, e	Jika Z=1 PC ← PC + e	Instruksi ini memungkinkan percabagang (loncatan) tak bersyarat ke segmen lain dari suatu program tergantung dari hasil *test Flag Zero*. Jika *Flag* = 1, nilai dari bilangan e ditambahkan ke *program counter* (PC) dan instruksi berikutnya diambil dari lokasi yang ditunjukkan oleh PC yang baru. Loncatan ini dihitung dari *address opcode* instruksi tersebut dan berada diantara -126 sampai +129 byte. *Assembler* secara otomatis menyesuaikan dengan cara mengurangi PC dengan 2.
JR NZ, e	Jika Z=0 PC ← PC + e	Instruksi ini memungkinkan percabagang (loncatan) tak bersyarat

		ke segmen lain dari suatu program tergantung dari hasil *test Flag Zero*. Jika *Flag* = 0, nilai dari bilangan e ditambahkan ke *program counter* (PC) dan instruksi berikutnya diambil dari lokasi yang ditunjukkan oleh PC yang baru. Loncatan ini dihitung dari *address opcode* instruksi tersebut dan berada diantara -126 sampai +129 byte. *Assembler* secara otomatis menyesuaikan dengan cara mengurangi PC dengan 2.
JP (HL)	PC ← HL	*Program counter* (pasangan *register* PC) diisi dengan isi pasangan register HL. Instruksi berikutnya diambil dari lokasi yang ditunjuk oleh isi PC yang baru.
JP (IX)	PC ← IX	*Program counter* (pasangan *register* PC) diisi dengan isi pasangan *register* indeks IX. Instruksi berikutnya diambil dari lokasi yang ditunjuk oleh isi PC yang baru.
JP (IY)	PC ← IY	*Program counter* (pasangan *register* PC) diisi dengan isi pasangan *register* indeks IY. Instruksi berikutnya diambil dari lokasi yang ditunjuk oleh isi PC yang baru.
DJNZ e	---	Instruksi ini hampir sama dengan instruksi percabangan (loncatan) bersyarat kecuali suatu nilai *register* digunakan untuk menentukan percabangannya. *Register* B dikurangi dengan 1 (*decremented*) dan apabila nilainya tidak sama dengan 0, nilai bilangan e ditambahkan ke PC dan instruksi berikutnya diambil dari lokasi

		yang ditunjukkan oleh PC yang baru. Loncatan ini dihitung dari *address opcode* instruksi tersebut dan berada diantara -126 sampai +129 byte. *Assembler* secara otomatis menyesuaikan dengan cara mengurangi PC dengan 2.

Contoh :

1. Jika *Flag Carry* (Flag C pada *register* F) di-set dan isi alamat 1520H adalah 03H, setelah pelaksanaan JP C, 1520H

   Program counter akan berisi 1520H, pada perioda mesin berikutnya CPU akan mengambil byte 03H dari alamat 1520H.

Tabel 10. Kelompok *Call, return* dan *restart*

Mnemonic	**Operasi**	**Keterangan**
RETI		Instruksi ini digunakan pada suatu akhir ‚*interrupt service routine*' untuk : 1. menyiapkan isi PC (analog dengan instruksi RET) 2. untuk memberi sinyal peralatan I/O bahwa *routine interrupt* telah selesai. Instruksi RETI memberikan fasilitas kepada peralatan dengan prioritas lebih tinggi untuk menunda *service routine* peralatan yang mempunyai prioritas lebih rendah. Instruksi ini juga mereset flip-flop IFF1 dan IFF2.

RETN		Digunakan pada akhir suatu '*service routine*' untuk *non maskable interrupt* ini melaksanakan *return* tak bersyarat yang fungsinya identik dengan instruksi RET. Yaitu : isi *program counter* (PC) yang telah disimpan diambil kembali dari bagian atas *stack* memori luar ; byte PC berorde rendah diisi dengan isi memori yang lokasinya ditunjukkan oleh *stack pointer* (SP), SP ditambah dengan 1, byte PC berorde tinggi diisi dengan isi memori yang lokasinya ditunjukkan oleh *Stack pointer* (SP), SP ditambah lagi dengan 1. kontrol sekarang dikembalikan pada *flow program* asal ; pada perioda mesin berikutnya CPU akan mengambil *opcode* berikutnya dari memori yang lokasinya ditunjukkan oleh PC. Keadaan IFF2 dicopy kembali ke IFF1 pada keadaan sebelum diterimanya NMI.
RST p	(SP-1) ← PC_T, (SP-2) ← PC_R, PC_T ← 0, PC_R ← P	Isi PC yang ada disimpan pada *stack* memori luar, dan pada lokasi halaman nol yang diberikan oleh operand p yang diisikan ke PC. Pelaksanaan program dimulai dengan opcode pada *address* yang sekarang ditunjuk oleh PC. Penyimpanan ini dimulai dengan cara ; pertama mengurangi dengan satu isi SP, mengisikan byte PC berorde tinggi ke memori yang addressnya ditunjuk SP, SP dikurangi lagi dengan 1, dan byte PC yang berorde rendah diisi dengan memori yang addressnya ditunjuk oleh SP yang sekarang.

Contoh :

1. Jika SP adalah 1000H dan isi PC adalah 1A45H, ketika sinyal *non maskable interrupt* (NMI) diterima, CPU akan mengabaikan instruksi berikutnya dan akan kembali ke memori pada alamat 0066H. isi PC sekarang (1A45H) akan disimpan ke stack alamat luar 0FFFH dan 0FFEH, byte berode tinggi dahulu dan 0066H akan diisikan ke PC. Alamat tersebut dimulai dengan '*interrupt service routine*' yang berakhir dengan instruksi RETN.

2. Jika isi PC adalah 15B3H, setelah pelaksanaan RST 18 H

   PC akan berisi 0018H, sebagai alamat opcode berikutnya yang akan diambil.

Tabel 11. Operasi – operasi input dan output

Mnemonic	**Operasi**	**Keterangan**
IN A, (n)	A ← (n)	Operand n ditempatkan pada 8 bit yang berorde rendah (A0 sampai A7) dari *address bus* yang memilih peralatan I/O pada salah satu dari 256 *port* yang dimungkinkan. Isi akumulator juga tampak pada 8 bit yang berorde tinggi (A8 sampai A15) dari *address bus* pada saat yang bersamaan. Kemudian satu byte dari *port* yang terpilih ditempatkan pada *data bus* dan dimasukan pada akumulator (*register* A) dalam CPU.
IN r, (c)	A ← (c)	Operand c ditempatkan pada 8 bit yang berorde rendah (A0 sampai A7)

		dari *address bus* yang memilih peralatan I/O pada salah satu dari 256 *port* yang dimungkinkan. Isi *register* B ditempatkan pada 8 bit yang berorde tinggi (A8 sampai A15) dari *address bus* pada saat yang bersamaan. Kemudian satu byte dari *port* yang terpilih ditempatkan pada data bus dan dimasukan pada register r dalam CPU.
INI	(HL) ← (c), B ← B-1, HL ← HL+1	Isi *register* C ditempatkan pada 8 bit yang berorde rendah (A0 sampai A7) dari *address bus* untuk memilih peralatan I/O pada salah satu dari 256 port yang dimungkinkan. *Register* B dapat digunakan sebagai *byte counter*, dan isinya ditempatkan pada 8 bit yang berorde tinggi (A8 sampai A15) dari *address bus* pada saat bersamaan. Kemudian satu byte dari *port* yang terpilih ditempatkan pada data bus dan dimasukan dalam CPU. Isi pasangan *register* HL kemudian ditempatkan pada *address bus* dan byte input dimasukan pada memori yang lokasinya ditunjukkan oleh isi HL. Akhirnya *byte counter* dkurangi dengan satu dan pasangan *register* HL ditambah dengan satu.
INIR	(HL) ← (c), B ← B-1, HL ← HL+1	Isi *register* C ditempatkan pada 8 bit yang berorde rendah (A0 sampai A7) dari *address bus* untuk memilih peralatan I/O pada salah satu dari 256 port yang dimungkinkan. *Register* B dapat digunakan sebagai *byte counter*, dan isinya ditempatkan pada 8 bit yang berorde tinggi (A8 sampai

		A15) dari *address bus* pada saat bersamaan. Kemudian satu byte dari *port* yang terpilih ditempatkan pada data bus dan dimasukan dalam CPU. Isi pasangan *register* HL kemudian ditempatkan pada *address bus* dan byte input dimasukan pada memori yang lokasinya ditunjukkan oleh isi HL. Lalu pasangan *register* HL ditambah dengan satu dan *byte counter* dikurangi dengan satu, instruksi dianggap selesai jika B belum sama dengan nol. PC dikurangi dengan dua dan instruksi diulangi. Catatan : jika B diset nol sebelum pelaksanaan instruksi ini, maka instruksi ini akan dilaksanakan sebanyak 256 kali. Interupsi akan dilayani setiap kali selesai mentranfer satu data.
IND	(HL) ← (c), B ← B-1, HL ← HL-1	Isi *register* C ditempatkan pada 8 bit yang berorde rendah (A0 sampai A7) dari address bus untuk memilih peralatan I/O pada salah satu dari 256 *port* yang dimungkinkan. *Register* B dapat digunakan sebagai byte *counter*, dan isinya ditempatkan pada 8 bit yang berorde tinggi (A8 sampai A15) dari *address bus* pada saat bersamaan. Kemudian satu byte dari *port* yang terpilih ditempatkan pada *data bus* dan byte input dari *port* yang terpilih ditempatkan pada data bus dan byte input dimasukan memori yang lokasinya ditunjukkan oleh HL. Akhirnya pasangan byte *counter* dan pasangan *register* HL dikurangi

		dengan satu.
INDR	(HL) ← (c), B ← B-1, HL ← HL-1	Isi *register* C ditempatkan pada 8 bit yang berorde rendah (A0 sampai A7) dari *address bus* untuk memilih peralatan I/O pada salah satu dari 256 port yang dimungkinkan. *Register* B dapat digunakan sebagai byte *counter*, dan isinya ditempatkan pada 8 bit yang berorde tinggi (A8 sampai A15) dari *address bus* pada saat bersamaan. Kemudian satu byte dari *port* yang terpilih ditempatkan pada *data bus* dan dimasukan dalam CPU. Isi pasangan *register* HL kemudian ditempatkan pada *address bus* dan byte input dimasukan pada memori yang lokasinya ditunjukkan oleh isi HL. Lalu pasangan *register* HL byte *counter* dikurangi dengan satu jika pengurangan itu menyebabkan B menjadi 0, instruksi dianggap selesai. Jika B belum sama dengan 0, PC dikurangi dengan dua dan instruksi diulangi. Catatan : jika B diset 0 sebelum pelaksanaan instruksi ini, maka instruksi ini akan dilaksanakan sebanyak 256 kali. Interupsi akan dilayani setiap kali selesai mentransfer data.
OUT (n), A	(n) ← A	*Operand* n ditempatkan pada 8 bit yang berorde rendah dari address bus untuk memilih peralatan I/O pada salah satu dari 256 port yang dimungkinkan. Isi akumulator juga tampak pada 8 bit yang berorde tinggi address bus pada saat yang bersamaan. Kemudian byte yang

		terdapat pada akumulator ditempatkan pada data bus dan dimasukan pada peralatan periperal yang terpilih.
OUT (c), r	(c) ← r	Isi *register* C ditempatkan pada 8 bit yang berorde rendah dari *address bus* untuk memilih peralatan I/O pada salah satu dari 256 port yang dimungkinkan. Isi akumulator juga tampak pada 8 bit yang berorde tinggi *address bus* pada saat yang bersamaan. Kemudian byte yang terdapat pada register r ditempatkan pada data bus dan dimasukan pada peralatan periperal yang terpilih.
OUTI	(c) ← (HL), B ← B-1, HL ← HL+1	Isi pasangan register HL ditempatkan pada *address bus* untuk memilih suatu lokasi pada memori. Byte yang berada lokasi ini disimpan sementara dalam CPU, lalu setelah byte *counter* (penghitung jumlah byte = *register* B) dikurangi dengan satu. Isi *register* c ditempatkan pada 8 bit berorde rendah dari *address bus* yang memilih peralatan I/O pada salah satu 256 *port* yang dimungkinkan. *Register* B dapat digunakan sebagai byte *counter* dan isinya yang telah dikurangi dengan satu ditempatkan pada 8 bit berorde tinggi dari *address bus*. Byte untuk output ditempatkan pada data bus dan dituliskan pada peralatan periperal yang terpilih. Akhirnya pasangan register HL ditambah dengan satu.
OTIR	(c) ← (HL), B ← B-1, HL	Isi pasangan *register* HL ditempatkan pada *address bus* untuk memilih

	← HL+1	suatu lokasi pada memori. Byte yang berada lokasi ini disimpan sementara dalam CPU, lalu setelah byte *counter* (penghitung jumlah byte = *register* B) dikurangi dengan satu. Isi *register* c ditempatkan pada 8 bit berorde rendah dari *address bus* yang memilih peralatan I/O pada salah satu 256 *port* yang dimungkinkan. *Register* B dapat digunakan sebagai byte *counter* dan isinya yang telah dikurangi dengan satu ditempatkan pada 8 bit berorde tinggi dari *address* bus. Byte untuk output ditempatkan pada data bus dan dituliskan pada peralatan periperal yang terpilih. Akhirnya pasangan *register* HL ditambah dengan satu. Jika pengurangan B dengan satu tersebut menyebabkan B menjadi 0, PC dikurangi dengan 2 dan instruksi diulangi. Catatan : jika B diset nol sebelum pelaksanaan instruksi ini, maka instruksi ini akan dilaksanakan sebanyak 256 kali. Interupsi akan dilayani setiap kali selesai mentransfer satu data.
OUTD	(c) ← (HL), B ← B-1, HL ← HL-1	Isi pasangan *register* HL ditempatkan pada *address bus* untuk memilih suatu lokasi pada memori. Byte yang berada lokasi ini disimpan sementara dalam CPU, lalu setelah byte *counter* (penghitung jumlah byte = *register* B) dikurangi dengan satu. Isi register c ditempatkan pada 8 bit berorde rendah dari *address bus* yang memilih peralatan I/O pada salah satu 256 port yang dimungkinkan. *Register* B dapat digunakan sebagai byte *counter* dan

		isinya yang telah dikurangi dengan satu ditempatkan pada 8 bit berorde tinggi dari address bus. Byte untuk output ditempatkan pada data bus dan dituliskan pada peralatan periperal yang terpilih. Akhirnya pasangan register HL dikurangi dengan satu.
OTDR	(c) ← (HL), B ← B-1, HL ← HL-1	Isi pasangan register HL ditempatkan pada *address bus* untuk memilih suatu lokasi pada memori. Byte yang berada lokasi ini disimpan sementara dalam CPU, lalu setelah byte *counter* (penghitung jumlah byte = *register* B) dikurangi dengan satu. Isi *register* c ditempatkan pada 8 bit berorde rendah dari *address bus* yang memilih peralatan I/O pada salah satu 256 *port* yang dimungkinkan. *Register* B dapat digunakan sebagai *byte counter* dan isinya yang telah dikurangi dengan satu ditempatkan pada 8 bit berorde tinggi dari *address bus*. Byte untuk *output* ditempatkan pada data bus dan dituliskan pada peralatan periperal yang terpilih. Akhirnya pasangan *register* HL dikurangi dengan satu. Jika pengurangan B dengan satu ini menyebabkan B menjadi 0, maka instruksi dianggap selesai . jika B belum sama dengan 0 PC dikurangi dengan 2 dan instruksi diulangi. Catatan : jika B diset nol sebelum pelaksanaan instruksi ini, maka instruksi ini akan dilaksanakan sebanyak 256 kali. Interupsi akan dilayani setiap kali selesai mentransfer satu data.

3. **Referensi :**

a. Harry Garland. (1979). *Introduction to microprocessor system design.*New Jersey : MC Graw Hill.

b. Laventhal, (1985). *Instroduction to microprocessor; software, hardware, programming.* Prentice Hall.

c. Laventhal, (1986). *Z80 Assembly Language Programming,* Mc Graw Hill, Singapore.

d. Inelco, (1986). *Guru Mikro Saya.*