Chapter 6

Programming the Basic Computer – Part 1

Based on slides by:
Prof. Myung-Eui Lee
Korea University of Technology & Education
Department of Information & Communication

Alon Schclar, Tel-Aviv College, 2009

Levels of Representation

High Level Language Program

Compiler

Assembly Language Program

Assembler

Machine Language Program

Machine Interpretation

Control Signal Specification

°
°

temp = v[k];
v[k] = v[k+1];
v[k+1] = temp;

lw $15, 0($2)
lw $16, 4($2)
sw $16, 0($2)
sw $15, 4($2)

0000 1001 1100 0110 1010 1111 0101 1000
1010 1111 0101 1000 0000 1001 1100 0110
1100 0110 1010 1111 0101 1000 0000 1001
0101 1000 0000 1001 1100 0110 1010 1111

DR $\leftarrow$ MEM[AC]

Talking to the Computer

- In order to "**talk**" to the computer we must send it electronic signals.
- The easiest signals for an electronic machine to understand are ***on*** and ***off***
  - Correspond to *high* voltage and *low* **voltage**.
- Thus the computer's alphabet is composed of two symbols **0** and **1**.
- Any "words" composed of these 2 numbers are called ***binary numbers***.

Talking to the Computer

- A computer needs **our instructions** in order to function
- Instructions are formulated as **binary** numbers
- The binary number 11101100100001 can be an instruction to **subtract** two numbers
- Every computer understands a predefined set of instructions – **instruction set**
- An instruction has a predefined **structure** which matches the architecture of the computer

5-3 Computer Instruction

3 Instruction Code Formats : *Fig. 5-5*

- Memory-reference instruction
  - » Opcode = 000 ~ 110
    - I=0 : 0xxx ~ 6xxx, I=1: 8xxx ~Exxx

I=0 : Direct,
I=1 : Indirect

15	14 12	11 0
I	Opcode	Address

- Register-reference instruction
  - » 7xxx (7800 ~ 7001) : CLA, CMA, ….

15	14 12	11 0
0	1 1 1	Register Operation

- Input-Output instruction
  - » Fxxx(F800 ~ F040) : INP, OUT, ION, SKI, ….

15	14 12	11 0
1	1 1 1	I/O Operation

Symbol	Hex Code I = 0	Hex Code I = 1	Description
AND	0xxx	8xxx	And memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and Save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME	7100		Comp m e
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrup
IOF	F040		Inter

TABLE 6-2 Binary Program to Add Two Numbers

Location	Instruction code
0	0010 0000 0000 0100
1	0001 0000 0000 0101
10	0011 0000 0000 0110
11	0111 0000 0000 0001
100	0000 0000 0101 0011
101	1111 1111 1110 1001
110	0000 0000 0000 0000

taken from **M. Mano/Computer Design and Architecture 3rd Ed.**

TABLE 6-3 Hexadecimal Program to Add Two Numbers

Location	Instruction
000	2004
001	1005
002	3006
003	7001
004	0053
005	FFE9
006	0000

taken from M. Mano/Computer Design and Architecture 3rd Ed.

Assembly Language

- Initially (in the late 40s) computer programs where written as binary numbers
  - They were input to the computer by **turning on and off** switches
- Although intriguing at first, it becomes extremely **tedious** after a short while
- Nest step: create **codes** for the instructions
  - Provides one level of abstraction
- **Convert** the code into binary
  - Manually at first

TABLE 6-4 Program with Symbolic Operation Codes

Location	Instruction	Comments
000	LDA 004	Load first operand into *AC*
001	ADD 005	Add second operand to *AC*
002	STA 006	Store sum in location 006
003	HLT	Halt computer
004	0053	First operand
005	FFE9	Second operand (negative)
006	0000	Store sum here

taken from M. Mano/Computer Design and Architecture 3rd Ed.

TABLE 6-5 Assembly Language Program to Add Two Numbers

	ORG 0	/Origin of program is location 0
	LDA A	/Load operand from location *A*
	ADD B	/Add operand from location *B*
	STA C	/Store sum in location *C*
	HLT	/Halt computer
A,	DEC 83	/Decimal operand
B,	DEC −23	/Decimal operand
C,	DEC 0	/Sum stored in location *C*
	END	/End of symbolic program

taken from M. Mano/Computer Design and Architecture 3rd Ed.

Assembly Language

- Next step: **automatic conversion** of the codes into binary instructions
- This program is called an ***assembler*.**
- The symbolic names of the instructions are called the *assembly language*.
- Assembly language increases productivity However,
  - Each machine instruction must be written on a single line
  - The programmer needs to **think like a machine**
- **A higher level of abstraction is needed**

High Level Languages

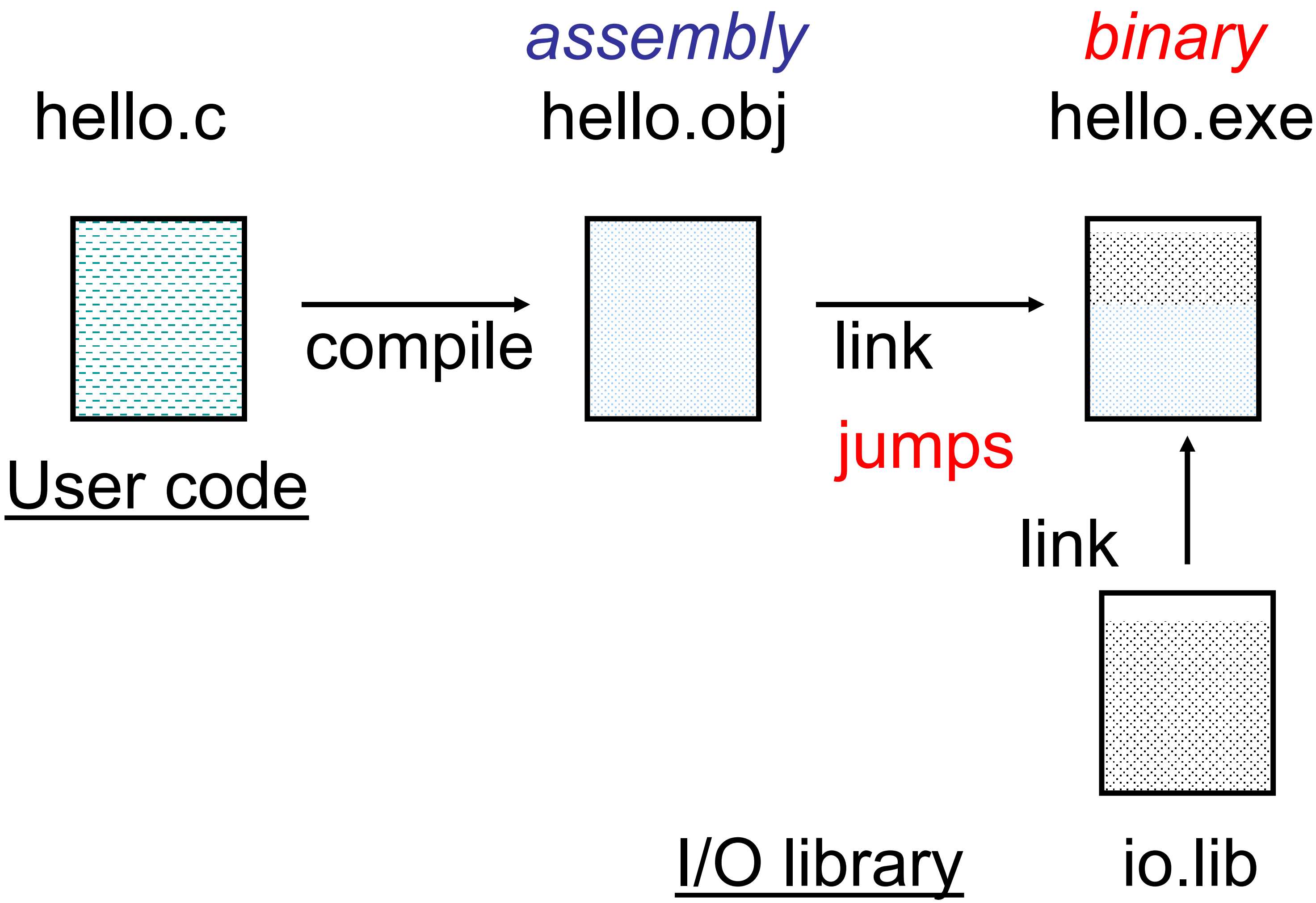
- Humans think in a language that consists of **sentences** over the alphabet
- Why not define a language that is more **intuitive** for the programmer
- Next step:
  - Define a **high level language** (JAVA, C)
  - Construct a translator that converts high level instructions into assembly – the **compiler**
  - From here we already know how to convert **assembly** to binary instructions

TABLE 6-6 Fortran Program to Add Two Numbers

```
INTEGER A, B, C
DATA A,83  B,-23
C = A + B
END
```

taken from M. Mano/Computer Design and Architecture 3rd Ed.

Compiling and Linking a Program



Executable program

- A sequence of machine instructions
  - Binary coded
  - Operations, operands (values)
  - Operands may be
    - Values – floating point, 2's-comp
    - Register numbers

From EXE file to execution

- Given a binary executable file
  - the file is ***loaded*** from the I/O device (hard disk, DVD, Disk-on-Key etc.) into the memory
  - the processor **executes** the instructions in an **iterative** process
  - the **operating system** coordinates this process

Chap. 6 Programming the Basic Computer

6-1 Introduction

- Translate user-oriented ***symbolic program***(alphanumeric character set) into ***binary programs*** recognized by the hardware
- 25 Instruction Set of the basic computer
  - Memory Reference Instruction
  - Register Reference Instruction
  - Input-output Instruction

6-2 Machine Language

- Program Categories
  - 1) Binary Code(Machine Language)
    - » Program Memory ***Tab. 6-2***
  - 2) Octal or Hexadecimal Code
    - » Binary Code ***Tab. 6-3***
  - 3) Symbolic Code : ***Tab. 6-4***
    - » Assembly Language : ***Tab. 6-5***
  - 4) High Level Language
    - » C, Fortran,.. : ***Tab 6-6***

Symbol	Hex Code		Description
AND	0xxx	8xxx	And memory word to AC
ADD	1xxx	9xxx	Add memory word to AC
LDA	2xxx	Axxx	Load memory word to AC
STA	3xxx	Bxxx	Store content of AC in memory
BUN	4xxx	Cxxx	Branch unconditionally
BSA	5xxx	Dxxx	Branch and Save return address
ISZ	6xxx	Exxx	Increment and skip if zero
CLA	7800		Clear AC
CLE	7400		Clear E
CMA	7200		Complement AC
CME e	7100		Comp
CIR	7080		Circulate right AC and E
CIL	7040		Circulate left AC and E
INC	7020		Increment AC
SPA	7010		Skip next instruction if AC positive
SNA	7008		Skip next instruction if AC negative
SZA	7004		Skip next instruction if AC zero
SZE	7002		Skip next instruction if E is 0
HLT	7001		Halt computer
INP	F800		Input character to AC
OUT	F400		Output character from AC
SKI	F200		Skip on input flag
SKO	F100		Skip on output flag
ION	F080		Interrup
IOF	F040		Inter

6-3 Assembly Language

The rules for writing assembly language program

- Documented and published in manuals(from the computer manufacturer)

Rules of the Assembly Language

- Each line of an assembly language program is arranged in three columns
  - » 1) Label field : empty or symbolic address
  - » 2) Instruction field : machine instruction or pseudoinstruction
  - » 3) Comment field : empty or comment

Field

Label	Instruction	Comment

- Symbolic Address(***Label field***)
  - » One, two, or three, but not more than three alphanumeric characters
  - » The first character must be a letter; the next two may be letters or numerals
  - » A symbolic address is terminated by a comma(*recognized as a label by the assembler*)
- Instruction Field
  - » 1) A memory-reference instruction(***MRI***)
    - Ex) **ADD OPR**(*direct address MRI*), **ADD PTR I**(*indirect address MRI*)
  - » 2) A register-reference or input-output instruction(***non-MRI***)
    - Ex) **CLA**(*register-reference*), **INP**(*input-output*)
  - » 3) A pseudoinstruction with(**ORG N**) or without(**END**) an operand : ***Tab. 6-7***

TABLE 6-7 Definition of Pseudoinstructions

Symbol	Information for the Assembler
ORG N	Hexadecimal number N is the memory location for the instruction or operand listed in the following line
END	Denotes the end of symbolic program
DEC N	Signed decimal number N to be converted to binary
HEX N	Hexadecimal number N to be converted to binary

taken from M. Mano/Computer Design and Architecture 3rd Ed.

- Comment field
  - » Comment filed must be preceded by a slash(*recognized by assembler as comment*)
- An Example Program : ***Tab. 6-8***
  - 83 - (- 23) = 83 + (2's Complement of -23)
    = 83 + 23
- Translation to Binary : ***Tab. 6-9***
  - Assembler = the translation of the **symbolic**(= ***assembly***) program into ***binary***
  - Address Symbol Table = Hexadecimal address of symbolic address
    - » MIN = 106, SUB = 107, DIF = 108
- Two Pass Assembler : *in next Sec. 6-4*
  - 1) 1st scan pass : generate user defined address symbol table
  - 2) 2nd scan pass : binary translation

Ex) LDA SUB
1) SUB = 107
2) 2107

6-4 The Assembler

- Source Program → Object Code → Binary Code

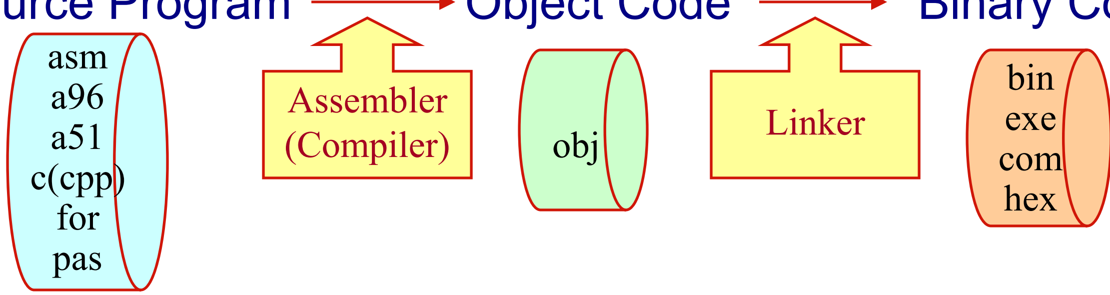


TABLE 6-8 Assembly Language Program to Subtract Two Numbers

	ORG 100	/Origin of program is location 100
	LDA SUB	/Load subtrahend to *AC*
	CMA	/Complement *AC*
	INC	/Increment *AC*
	ADD MIN	/Add minuend to *AC*
	STA DIF	/Store difference
	HLT	/Halt computer
MIN,	DEC 83	/Minuend
SUB,	DEC −23	/Subtrahend
DIF,	HEX 0	/Difference stored here
	END	/End of symbolic program

taken from M. Mano/Computer Design and Architecture 3rd Ed.

◆ Representation of Symbolic Program in Memory : *Tab. 6-11*

- Line of Code : **PL3, LDA SUB I** (Carriage return)
  - » The assembler recognizes a ***CR*** code as the end of a line of code

◆ Two Pass Assembler

- 1) ***1st pass*** : Generate user-defined ***address symbol table***
  - » Flowchart of first pass : ***Fig. 6-1***
  - » Address Symbol Table for Program in ***Tab. 6-8*** : ***Tab. 6-12***

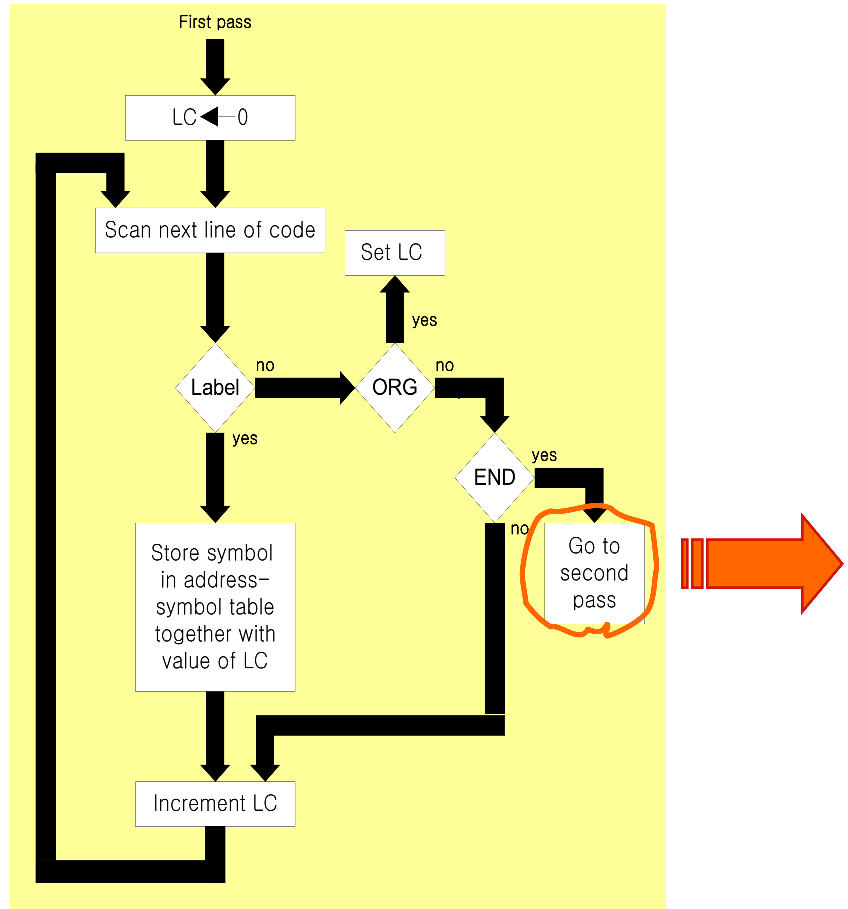


Fig. 6-1 Flowchart for first pass of assembler

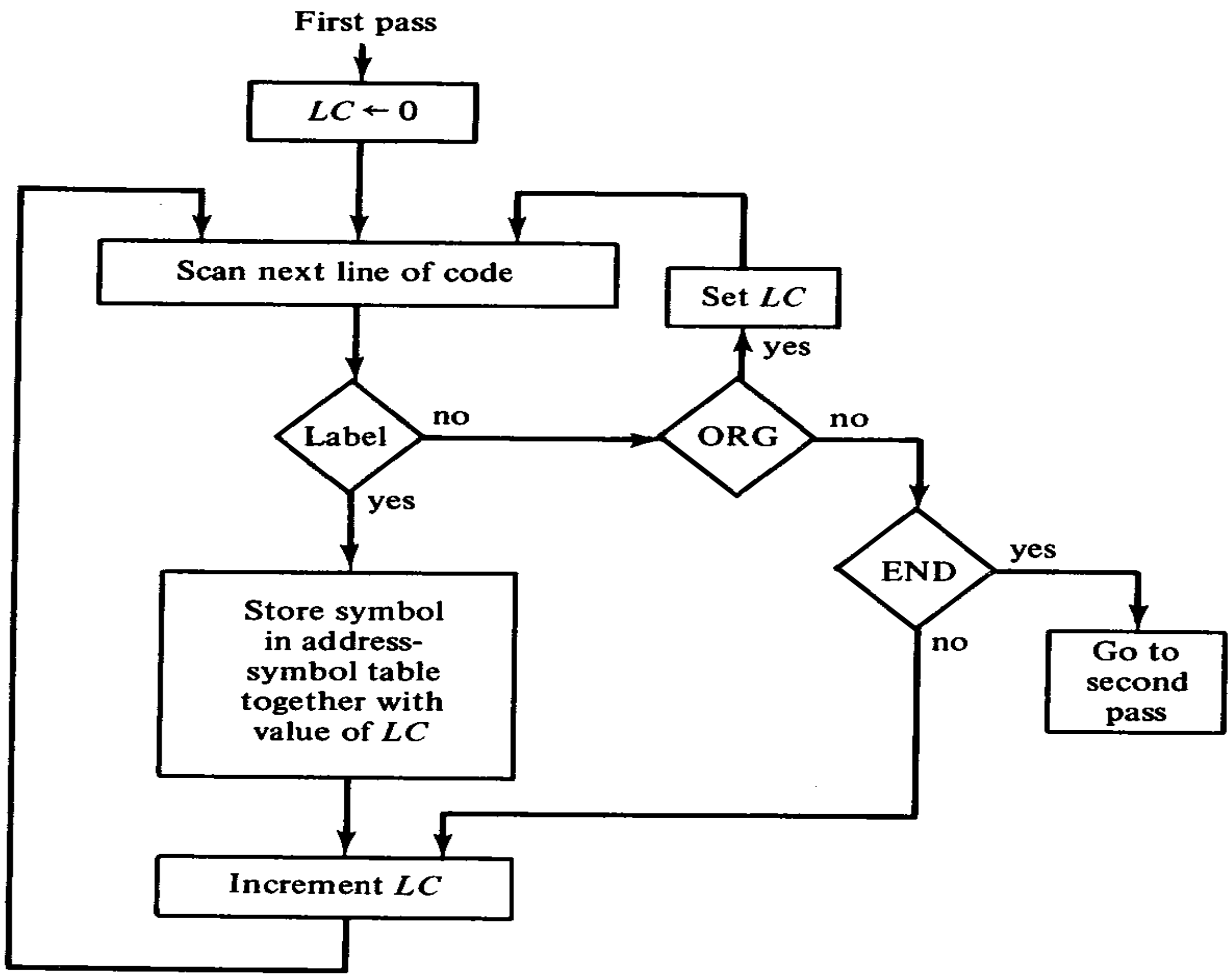


Figure 6-1 Flowchart for first pass of assembler.

Alon Schclar, Tel-Aviv College, 2009

taken from M. Mano/Computer Design and Architecture 3rd Ed.

Address symbol	Hexadecimal address
MIN	106
SUB	107
DIF	108

taken from M. Mano/Computer Design and Architecture 3rd Ed.

TABLE 6-12 Address Symbol Table for Program in Table 6-8

Memory word	Symbol or (LC)*	Hexadecimal code	Binary representation
1	M I	4D 49	0100 1101 0100 1001
2	N ,	4E 2C	0100 1110 0010 1100
3	(LC)	01 06	0000 0001 0000 0110
4	S U	53 55	0101 0011 0101 0101
5	B ,	42 2C	0100 0010 0010 1100
6	(LC)	01 07	0000 0001 0000 0111
7	D I	44 49	0100 0100 0100 1001
8	F ,	46 2C	0100 0110 0010 1100
9	(LC)	01 08	0000 0001 0000 1000

* (LC) designates content of location counter.

TABLE 6-10 Hexadecimal Character Code

Character	Code	Character	Code	Character	Code
A	41	Q	51	6	36
B	42	R	52	7	37
C	43	S	53	8	38
D	44	T	54	9	39
E	45	U	55	space	20
F	46	V	56	(	28
G	47	W	57	)	29
H	48	X	58	*	2A
I	49	Y	59	+	2B
J	4A	Z	5A	,	2C
K	4B	0	30	−	2D
L	4C	1	31	.	2E
M	4D	2	32	/	2F
N	4E	3	33	=	3D
O	4F	4	34	CR	0D (carriage return)
P	50	5	35		

taken from M. Mano/Computer Design and Architecture 3rd Ed.

- 2) ***2nd pass*** : Binary translation

  Instruction Format ***Binary Code*** (Pseudoinstruction Table, MRI Table, Non-MRI Table, Address Symbol Table)

  » Flowchart of second pass : ***Fig. 6-2***

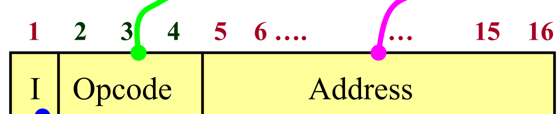
  Lookup tables

  » Binary Code translation

  Tab. 6-9 ***Content***

  » Error Diagnostics

  - Check for possible errors in the symbolic program
  - Ex) Invalid Machine Code Error



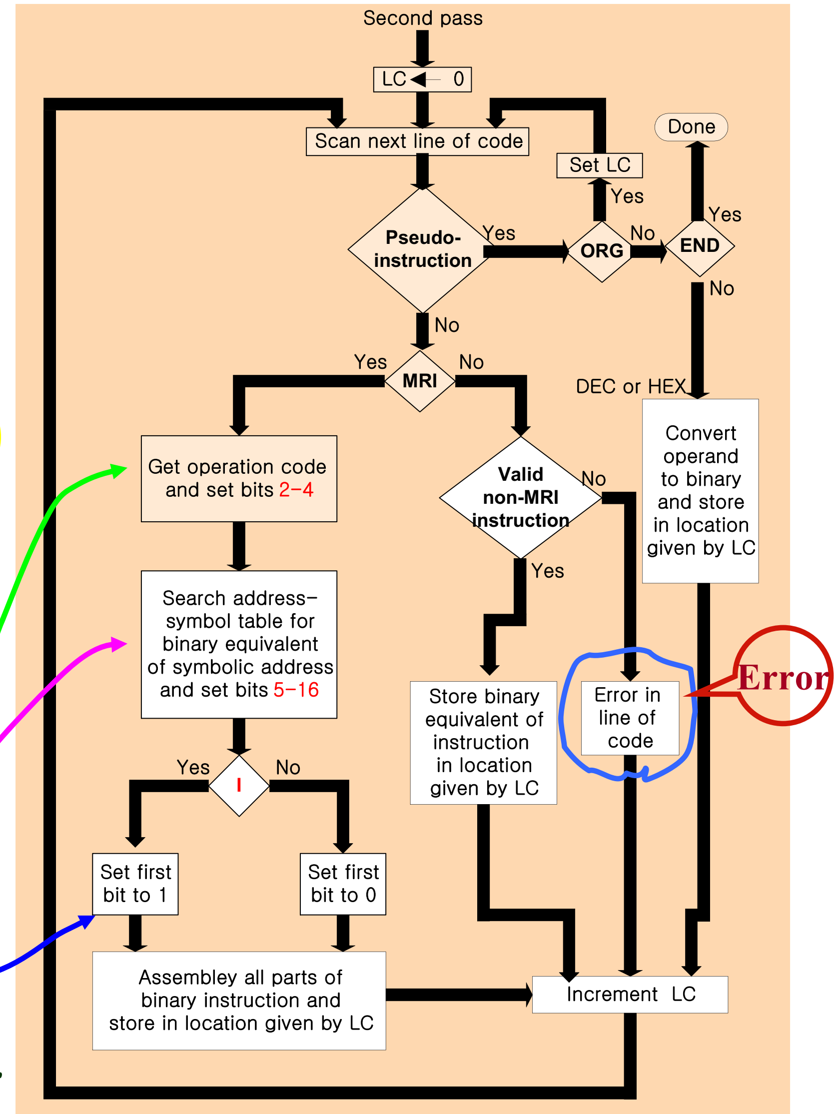


Fig. 6-2 Flowchart for second pass of assembler

Error diagnostics

- Invalid machine code symbol
  - Absent in the MRI and non-MRI tables.
  - The assembler cannot translate such a symbol
  - Prints an error message at a specific line of code.
- Symbolic address that did not appear as a label
  - The assembler cannot translate the line of code
  - No binary equivalent of the symbol in the symbol table generated during the first pass.

RUN THE EXAMPLE IN THE SIMULATOR
MICRO/MACRO MODE

TABLE 6-9 Listing of Translated Program of Table 6-8

Hexadecimal code			
Location	Content	Symbolic program	
			ORG 100
100	2107 Convert symbols		LDA SUB
101	7200		CMA
102	7020		INC
103	1106		ADD MIN
104	3108		STA DIF
105	7001		HLT
106	0053	MIN,	DEC 83
107	2's comp FFE9	SUB,	DEC −23
108	0000	DIF,	HEX 0
			END

Exercises – Chapter 6
Part 1

All questions are taken from Chapter 6 in
M. Mano/Computer Design and Architecture 3rd Ed.

Alon Schclar, Tel-Aviv College, 2009

Exercise 1

- The following program is stored in the memory unit of the basic computer.
- Show the contents of the ***AC, PC,*** and ***IR*** (*in hexadecimal*), after each instruction is executed.
- *All* numbers listed below are in *hexadecimal*.

Location	Instruction
010	CLA
011	ADD 016
012	BUN 014
013	HLT
014	AND 017
015	BUN 013
016	C1A5
017	93C6

Solution 1

		AC	PC	IR
010	CLA	0000	011	7800
011	ADD 016	C1A5	012	1016
012	BUN 014	C1A5	014	4014
013	HLT	8184	014	7001
014	AND 017	8184	015	0017
015	BUN 013	8184	013	4013
016	C1A5			
017	93C6			

$$(C1A5)_{16} = 1100\ 0001\ 1010\ 0101 \quad \text{AND}$$
$$(93C6)_{16} = \underline{1001\ 0011\ 1100\ 0110}$$
$$1000\ 0001\ 1000\ 0100 = (8184)_{16}$$

Exercise 2

- What happens during the first pass of the assembler if the line of code that has a pseudoinstruction ORG or END also has a label?
- Modify the flowchart to include an error message if this occurs.

Solution 2

- The assembler will not detect an ORG or END if the line has a label
- When this happens, an error needs to be issued

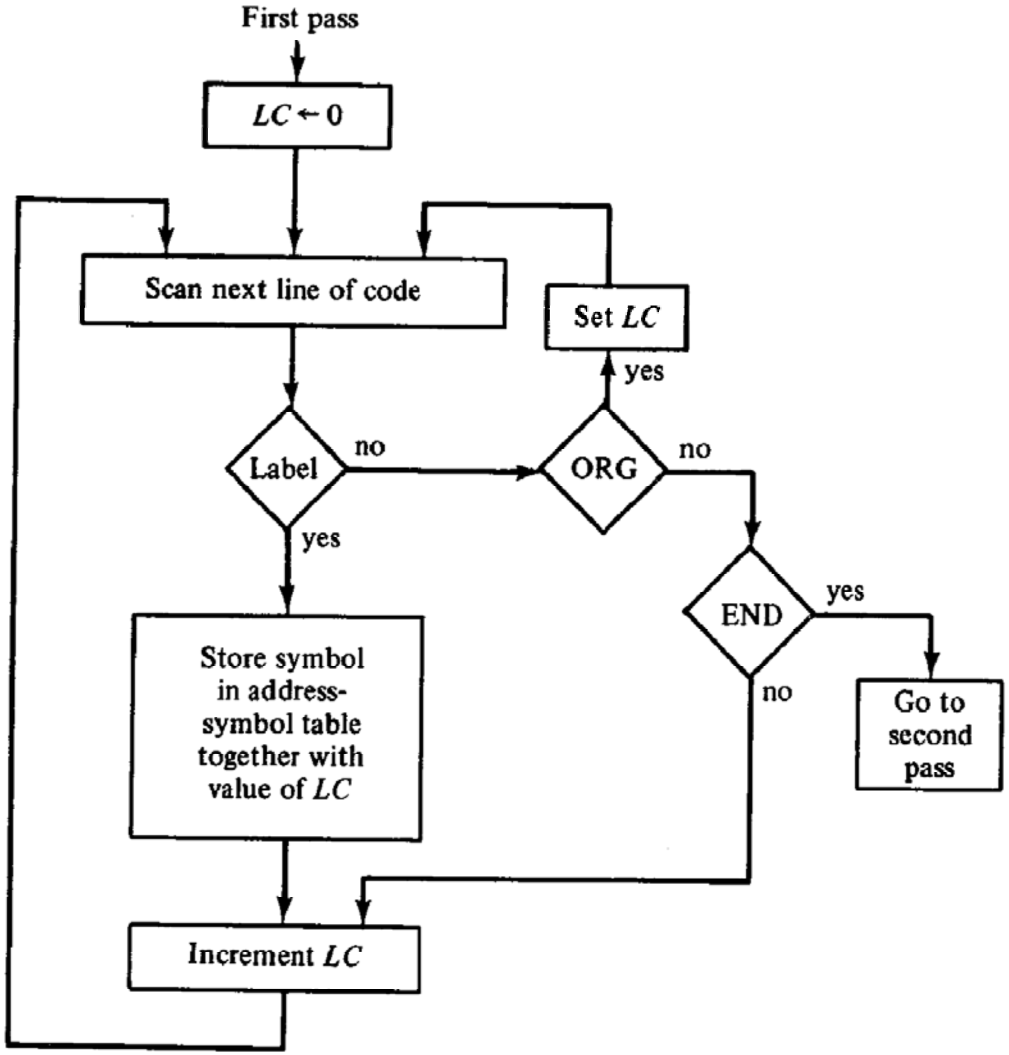
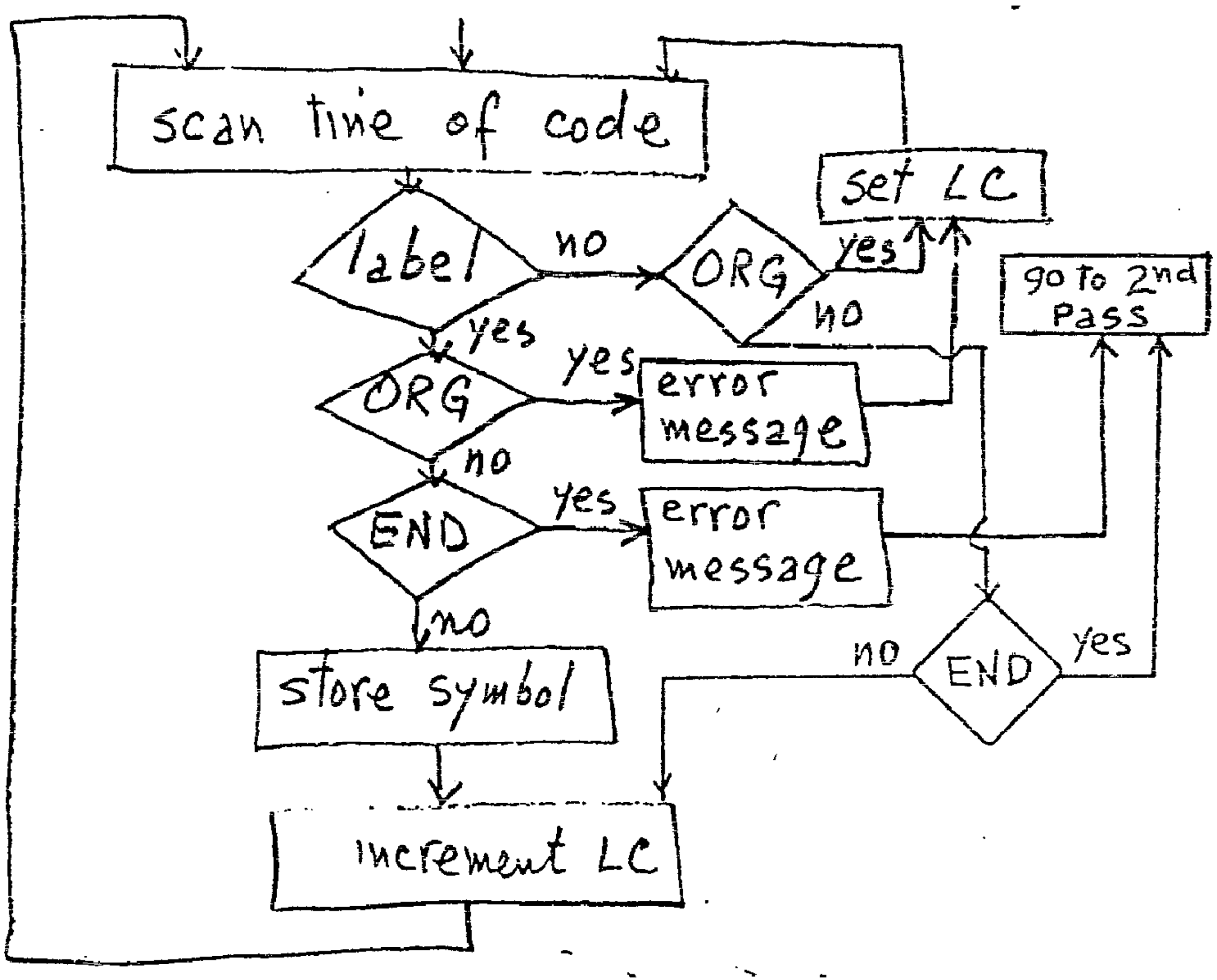


Figure 6-1 Flowchart for first pass of assembler.

Solution 2 – *cont.*



Exercise 3

- List the assembly language program (of the equivalent binary instructions) generated by a compiler for the following IF statement:

IF(A-B) 10, 20, 30

- The program branches to statement
  - 10 if A-B < 0;
  - 20 if A-B = 0; and
  - 30 if A-B > 0

Solution 3

```
LDA  B
CMA
INC
ADD  A     / Form A-B
SPA        / skip if AC Non Negative
BUN  N10   / (A-B)<0, go to N10
SZA        / Skip if AC=0
BUN  N30   / (A-B)>0, go to N30
BUN  N20   / (A-B)=0, go to N20
```

Chapter 6

Programming the Basic Computer – Part 2

Based on slides by:
Prof. Myung-Eui Lee
Korea University of Technology & Education
Department of Information & Communication

Alon Schclar, Tel-Aviv College, 2009

6-5 Program Loops

- **Program Loops**
  - A sequence of instructions that are executed many times
- **Example of program loop**
  - Sum of 100 integer numbers
    - » **Fortran**

```
   DIMENSION  A(100)
   INTEGER  SUM, A
   SUM = 0
   DO  3  J = 1, 100
3  SUM = SUM + A(J)
```

Tab. 6-13 Symbolic Program to Add 100 numbers

Line				
1		ORG	100	
2		LDA	ADS	/ A = 150
3		STA	PTR	/ PTR = 150
4		LDA	NBR	/ A = −100
5		STA	CTR	/ CTR = −100
6		CLA		/ A = 0
7	LOP,	ADD	PTR I	/ A + 75
8		ISZ	PTR	/ 150 + 1 = 151
9		ISZ	CTR	/ −100 + 1 = −99
10		BUN	LOP	/ Loop until CTR = 0
11		STA	SUM	/ Store A to SUM
12		HLT		
13	ADS,	HEX	150	
14	PTR,	HEX	0	/ 150
15	NBR,	DEC	−100	
16	CTR,	HEX	0	/ −100
17	SUM,	HEX	0	/ Result of Sum
18		ORG	150	
19		DEC	75	
,		,	,	Data
,		,	.	
118		DEC	23	
119		END		

TABLE 6-13 Symbolic Program to Add 100 Numbers

Line			
1		ORG 100	/Origin of program is HEX 100
2		LDA ADS	/Load first address of operands
3		STA PTR	/Store in pointer
4		LDA NBR	/Load minus 100
5		STA CTR	/Store in counter
6		CLA	/Clear accumulator
7	LOP,	ADD PTR I	/Add an operand to *AC*
8		ISZ PTR	/Increment pointer
9		ISZ CTR	/Increment counter
10		BUN LOP	/Repeat loop again
11		STA SUM	/Store sum
12		HLT	/Halt
13	ADS,	HEX 150	/First address of operands
14	PTR,	HEX 0	/This location reserved for a pointer
15	NBR,	DEC −100	/Constant to initialized counter
16	CTR,	HEX 0	/This location reserved for a counter
17	SUM,	HEX 0	/Sum is stored here
18		ORG 150	/Origin of operands is HEX 150
19		DEC 75	/First operand
•			
•			
•			
118		DEC 23	/Last operand
119		END	/End of symbolic program

Alon Schclar, Tel-Aviv College, 2009

taken from M. Mano/Computer Design and Architecture 3rd Ed.

Program to add two numbers

- Reserve 100 words of memory for **100** operands.
- The numbers are **integers**.
  - If they were of the **float** type,
    - compiler **reserves locations** for floating-point numbers
    - generate **instructions** that perform the subsequent **arithmetic** with **floating-point** data.
- **DIM** and **INTEGER** - **nonexecutable** statements similar assembly pseudoinstructions
- Suppose that the compiler reserves **locations $(150)_{16}$ to $(1B3)_{16}$** for the 100 operands.
  - These reserved memory words are listed in **lines 19 to 118**
  - Done by the **ORG pseudoinstruction** in line 18, which specifies the origin of the operands.

Program to add two numbers – cont.

- The **first** and **last** operands are listed with a specific **decimal number**
  - These **values** are **not known** during compilation.
  - Compiler **just reserves** the data space in memory
  - **Values** are **inserted later** when an **input** data **statement** (not listed in the program)
- Line numbers are for reference only
  - not part of the translated symbolic program.

Program to add two numbers – cont.

- **Line 9**: Only the increment part of ISZ is used
- **AC** is used for **SUM**
  - More efficient than to use a memory location
- **PTR**, **CTR** are **memory** words
  - When **more registers** are **available** (**RISC**) an intelligent compiler will **use registers**

6-6 Programming Arithmetic & Logic Operations

Hardware implementation

- Operations are implemented in a computer with one machine instruction
- Ex) ADD, AND

Software implementation

Hardware - faster and expensive
Software - slower and cheaper

- Operations are implemented by a set of instruction(Subroutine)
- Ex) **MUL**, **DIV**

Multiplication Program

- Positive Number Multiplication
  - » X = multiplicand
    Y = multiplier
    P = Partial Product Sum

```
X = 1 1 1 1
Y = 1 0 1 1
----------------
          1 1 1 1
        1 1 1 1
      0 0 0 0
    1 1 1 1
----------------
  1 0 1 0 0 1 0 1
```

```
P = 0 0 0 0 0 0 0 0
  + 0 0 0 0 1 1 1 1
P = 0 0 0 0 1 1 1 1
  + 0 0 0 1 1 1 1 0
P = 0 0 1 0 1 1 0 1
  + 0 0 0 0 0 0 0 0
P = 0 0 1 0 1 1 0 1
  + 0 1 1 1 1 0 0 0
P = 1 0 1 0 0 1 0 1
```

Algorithm *Fig. 6-3*

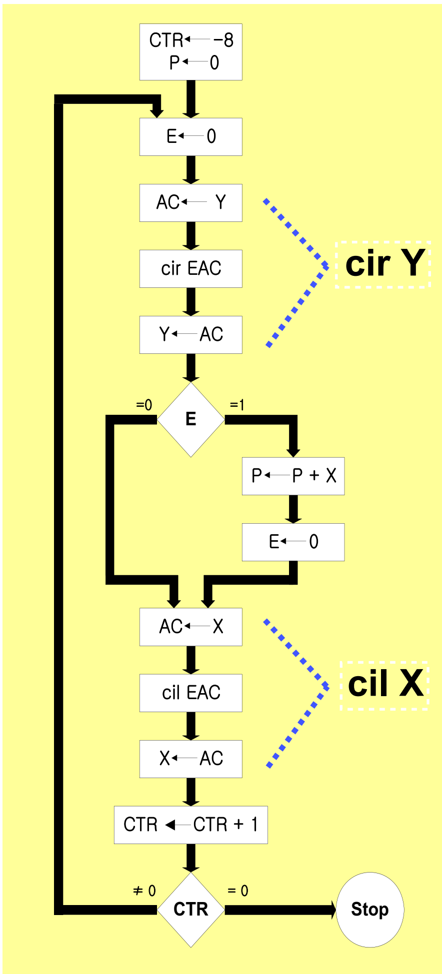
Circular Right

- E = 1
- E = 0

Multiplication program

- **Positive numbers** - disregard sign bit and
- No more than **eight significant** bits
  - their **product** cannot exceed the word capacity of **16 bits**.
  - for **16-bit numbers product** may be up to **31 bits** in length and will occupy two words of memory.
- Solution (like **pen** and **paper**)
  - checking the bits of the **multiplier Y**
  - adding the multiplicand **X** as many times as there are **1**'s in **Y**,
    - the value of **X** is shifted left from one line to the next.
- Reserve a memory location, **P**,
  - to store intermediate sums (partial products)
  - since computer can add only two numbers at a time,
  - **P** starts with zero

Fig. 6-3 flowchart for Multiplication Program



Tab. 6-14 Program to Multiply Two Positive numbers

Line				
1		ORG	100	
2	LOP,	CLE		/ A = 0
3		LDA	Y	/ A = Y (000B)
4		CIR		/ Circular Right to E
5		STA	Y	/ Store shifted Y
6		SZE		/ Check if E = 0
7		BUN	ONE	/ E = 1
8		BUN	ZRO	/ E = 0
9	ONE,	LDA	X	A = X (000F)
10		ADD	P	/ X = X + P
11		STA	P	/ St p
12		CLE		/ Clear E
13	ZRO,	LDA	X	/ A = X
14		CIL		/ A = 00011110 (00001111)
15		STA	X	/ St p
16		ISZ	CTR	/ CTR = − 7 = −8 + 1
17		BUN	LOP	/ Repeat until CTR = 0
18		HLT		
19	CTR,	DEC	−8	
20	X,	HEX	000F	
21	Y,	HEX	000B	
22	P,	HEX	0	
23		END		

Alternative?

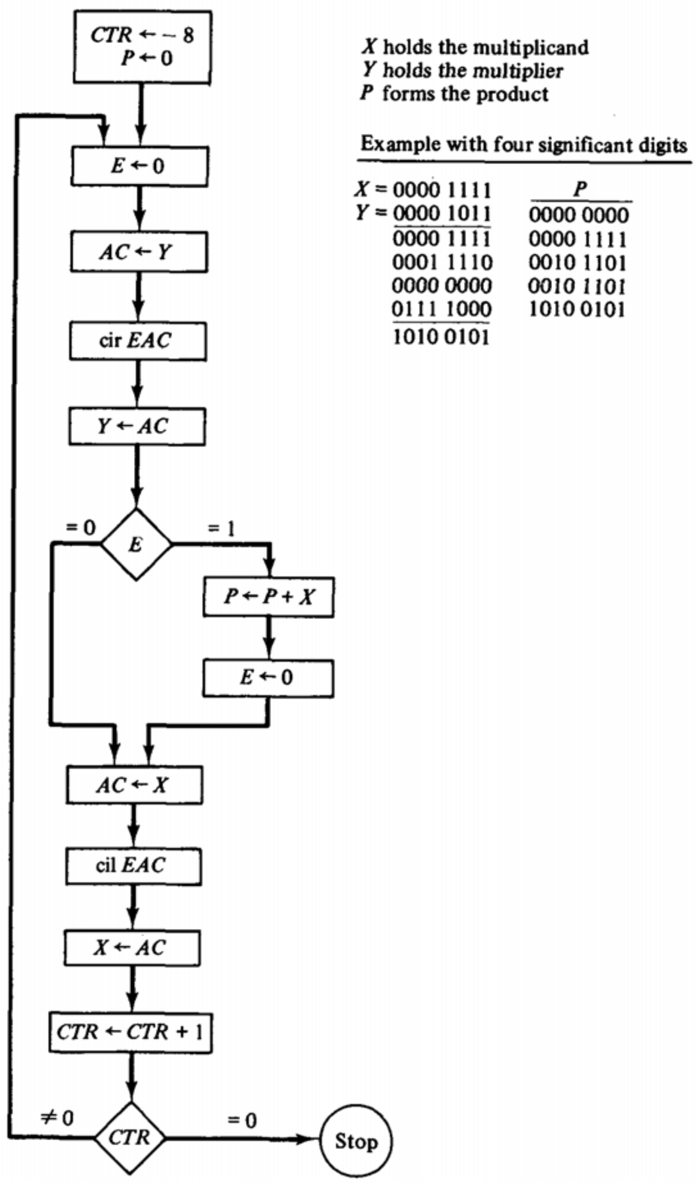


Figure 6-3 Flowchart for multiplication program.

TABLE 6-14 Program to Multiply Two Positive Numbers

	ORG 100	
LOP,	CLE	/Clear *E*
	LDA Y	/Load multiplier
	CIR	/Transfer multiplier bit to *E*
	STA Y	/Store shifted multiplier
	SZE	/Check if bit is zero
	BUN ONE	/Bit is one; go to ONE
	BUN ZRO	/Bit is zero; go to ZRO
ONE,	LDA X	/Load multiplicand
	ADD P	/Add to partial product
	STA P	/Store partial product
	CLE	/Clear *E*
ZRO,	LDA X	/Load multiplicand
	CIL	/Shift left
	STA X	/Store shifted multiplicand
	ISZ CTR	/Increment counter
	BUN LOP	/Counter not zero; repeat loop
	HLT	/Counter is zero; halt
CTR,	DEC −8	/This location serves as a counter
X,	HEX 000F	/Multiplicand stored here
Y,	HEX 000B	/Multiplier stored here
P,	HEX 0	/Product formed here
	END	

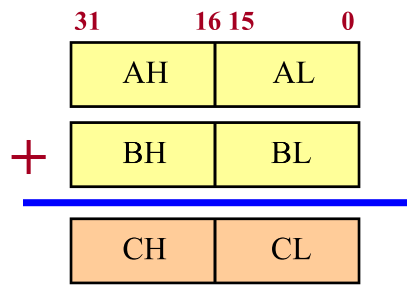
Alon Schclar, Tel-Aviv College, 2009

taken from M. Mano/Computer Design and Architecture 3rd Ed.

◆ Double Precision Addition : 32 bits

- **AL + BL**

E (**AH + BH + E**)



Line				
1		LDA	AL	/ A = AL
2		ADD	BL	/ A = AL + BL
3		STA	CL	/ Store A to CL
4		CLA		/ A = 0
5		CIL		/ 0000 0000 0000 000? (?=E)
6		ADD	AH	/ A = 00(E=0) or 01(E=1)
7		ADD	BH	/ A = A + AH + BH
8		STA	CH	/ Store A to CH
9		HLT		
10	AL,	DEC	?	/ Operand
11	AH,	DEC	?	
12	BL,	DEC	?	
13	BH,	DEC	?	
14	CL,	HEX	0	
15	CH,	HEX	0	

◆ Logic Operations

- Logic Operation

OR - How ? DeMorgan's law

» $A + B = \overline{\overline{A + B}} = \overline{\overline{A} \cdot \overline{B}}$

NAND - מערכות אוניברסליות

LDA	A		/ Load A
CMA			/ Complement A
STA	TMP	P	/ St ɒ
LDA	B		/ Load B
CMA			/ Complement
AND	TMP		/ AND
CMA			/ Complement

Program to Add Two Double-Precision Numbers

TABLE 6-15 Program to Add Two Double-Precision Numbers

```
        LDA AL    /Load A low
        ADD BL    /Add B low, carry in E
        STA CL    /Store in C low
        CLA       /Clear AC
        CIL       /Circulate to bring carry into AC(16)
        ADD AH    /Add A high and carry
        ADD BH    /Add B high
        STA CH    /Store in C high
        HLT
AL,     —         /Location of operands
AH,     —
BL,     —
BH,     —
CL,     —
CH,     —
```

◆ Shift Operations

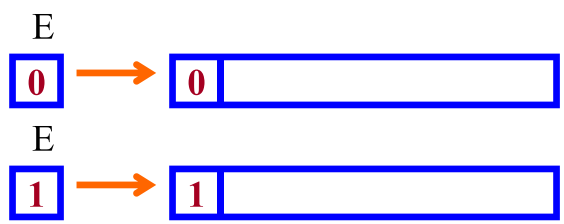
- Logical Shift : Zero must added to the extreme position
  - » Shift Right

    CLE
    CIR

  - » Shift Left

    CLE
    CIL

- Arithmetic Shift Right
  - » Positive (**+ = 0**)
  - » Negative (**- = 1**)



CLE	/ E= 0
SPA	/ Skip if A= +, E= 0
CME	/ Toggle E(=1) if A= −
CIR	/ Circulate A with E

■ 6-7 Subroutines

◆ Subroutine

- A set of common instruction that can be used in a program *many times*
- In basic computer, the link between the main program and a subroutine is the ***BSA*** instruction(*Branch and Save return Address*)
- Subroutine example : ***Tab. 6-16***

Location				
		ORG	100	/ Main Program
100		LDA	X	/ Load X
101		BSA	SH4	/ Call SH4 with X
102		STA	X	/ Store result X
103		LDA	Y	/ Load Y
104		BSA	SH4	/ Call SH4 with Y
105		STA	Y	/ Store result Y
106		HLT		
107	X,	HEX	1234	/ Result = 2340
108	Y,	HEX	4321	/ Result = 3210
				/ Subr
109	SH4,	HEX	0	/ Save Return Address
10A		CIL		
10B		CIL		
10C		CIL		
10D		CIL		
10E		AND	MSK	/ Mask lower 4 bit
10F		BUN	SH4 I	/ Indirect Return to main
110	MSK,	HEX	FFF0	/ Mask pattern
110		END		

Subroutine CALL hear

X = 102
Y = 105

Tab. 6-16 Program to Demonstrate the use of Subroutines

TABLE 6-16 Program to Demonstrate the Use of Subroutines

Location			
		ORG 100	/Main program
100		LDA X	/Load X
101		BSA SH4	/Branch to subroutine
102		STA X	/Store shifted number
103		LDA Y	/Load Y
104		BSA SH4	/Branch to subroutine again
105		STA Y	/Store shifted number
106		HLT	
107	X,	HEX 1234	
108	Y,	HEX 4321	
			/Subroutine to shift left 4 times
109	SH4,	HEX 0	/Store return address here
10A		CIL	/Circulate left once
10B		CIL	
10C		CIL	
10D		CIL	/Circulate left fourth time
10E		AND MSK	/Set *AC*(13–16) to zero
10F		BUN SH4 I	/Return to main program
110	MSK,	HEX FFF0	/Mask operand
		END	

Subroutine Parameters & Data Linkage

- Parameter(or Argument) Passing
  - » When a subroutine is called, the main program must transfer the data
- Parameter Passing
  - » 1) Data transfer through the ***Accumulator***
    - Used for only single input and single output parameter
  - » 2) Data transfer through the ***Memory***
    - Operand are often ***placed in memory locations following the CALL***
- 2 Parameter Passing ***Tab. 6-17***
  - » ***First Operand and Result*** : Accumulator
  - » ***Second Operand*** : Inserted in location following the ***BSA***
- BSA 2 Operand : ***Tab. 6-18***
  - » BSA 2 Operand
  - » Block Source Destination Address

Tab. 6-17 Program to Demonstrate Parameter Linkage

Location				
		ORG	200	
200		LDA	X	/ Load first operand X
201		BSA	OR	/ Call OR with X
202		HEX	3AF6	/ Second operand
203		STA	Y	/ Subroutine return here(Y=result)
204		HLT		
205	X,	HEX	7B95	/ First operand
206	Y,	HEX	0	/ Result store here
207	OR,	HEX	0	/ Return address = 202
208		CMA		/ Complement X
209		STA	TMP	/ TMP = X
20A		LDA	OR I	/ A = 3AF6 (202)
20B		CMA		/ Complement Second operand
20C		AND	TMP	/ AND
20D		CMA		/ Complement
20E		ISZ	OR	/ Return Address = 202 + 1 = 203
20F		BUN	OR I	/ Return to main
210	TMP,	HEX	0	
		END		

* OR Subroutine
First Operand : X = 7B95
Second Operand : BSA = 3AF6

TABLE 6-17 Program to Demonstrate Parameter Linkage

Location			
		ORG 200	
200		LDA X	/Load first operand into *AC*
201		BSA OR	/Branch to subroutine OR
202		HEX 3AF6	/Second operand stored here
203		STA Y	/Subroutine returns here
204		HLT	
205	X,	HEX 7B95	/First operand stored here
206	Y,	HEX 0	/Result stored here
207	OR,	HEX 0	/Subroutine OR
208		CMA	/Complement first operand
209		STA TMP	/Store in temporary location
20A		LDA OR I	/Load second operand
20B		CMA	/Complement second operand
20C		AND TMP	/AND complemented first operand
20D		CMA	/Complement again to get OR
20E		ISZ OR	/Increment return address
20F		BUN OR I	/Return to main program
210	TMP,	HEX 0	/Temporary storage
		END	

Tab. 6-18 Subroutine to Move a Block of Data

		ORG	100	
100		BSA	MVE	/ Subroutine Call
101		HEX	200	/ Source Address
102		HEX	300	/ Destin Addressa
103		DEC	−16	/ Number of data to move
104		HLT		
105	MVE,	HEX	0	/ Return address= 101
106		LDA	MVE I	/ A= 200
107		STA	PT1	/ PT1= 200
108		ISZ	MVE	/ Return address= 102
109		LDA	MVE I	/ A= 300
10A		STA	PT2	/ PT2= 300
10B		ISZ	MVE	/ Return address= 103
10C		LDA	MVE I	/ A= −16
10D		STA	CTR	/ CTR= −16
10E		ISZ	MVE	/ Return address= 104
10F	LOP,	LDA	PT1 I	/ A= Address 200
110		STA	PT2 I	/ Address 300
111		ISZ	PT1	/ PT1= 201
112		ISZ	PT2	/ PT2= 301
113		ISZ	CTR	/ CTR= −15 if 0 skip
114		BUN	LOP	/ Loop until CTR= 0
115		BUN	MVE I	/ 104 Retur = HLT
116	PT1,	HEX	?	/ Source
117	PT2,	HEX	?	/ Destination
118	CTR,	DEC	?	/ Counter

2 Operand

Subroutine to Move a **Block of** Data

TABLE 6-18 Subroutine to Move a Block of Data

		/Main program	
	BSA MVE	/Branch to subroutine	
	HEX 100	/First address of source data	
	HEX 200	/First address of destination data	3 parameters
	DEC −16	/Number of items to move	
	HLT		
MVE,	HEX 0	/Subroutine MVE	
	LDA MVE I	/Bring address of source	
	STA PT1	/Store in first pointer	
	ISZ MVE	/Increment return address	
	LDA MVE I	/Bring address of destination	
	STA PT2	/Store in second pointer	
	ISZ MVE	/Increment return address	
	LDA MVE I	/Bring number of items	
	STA CTR	/Store in counter	
	ISZ MVE	/Increment return address	
LOP,	LDA PT1 I	/Load source item	
	STA PT2 I	/Store in destination	
	ISZ PT1	/Increment source pointer	
	ISZ PT2	/Increment destination pointer	
	ISZ CTR	/Increment counter	
	BUN LOP	/Repeat 16 times	
	BUN MVE I	/Return to main program	
PT1,	—		
PT2,	—		
CTR,	—		

6-8 Input-Output Programming

One-character I/O

- Programmed I/O

Tab. 6-19 Program to input and output One character

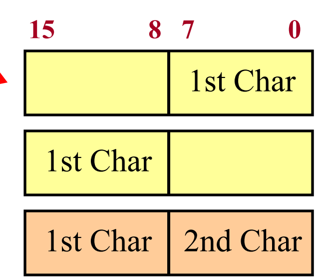
(a) Input a character			
CIF,	SKI		/ Check FGI = 1 ?
	BUN	CIF	/ Go to CIF if FGI= 0
	INP		/ Input character (FGI = 1)
	~~OUT~~		/ Echo Back
	STA	CHR	/ Store character
	HLT		
CHR,	?		/ Store character here
(b) Output a character			
	LDA	CHR	/ Load output character
COF,	SKO		/ Check FGO = 1 ?
	BUN	COF	/ Go to COF if FGO= 0
	OUT		/ Output character (FGO = 1)
	HLT		
CHR,	HEX	0057	/ Output character = "W"

Two-character I/O

- Two character Packing

Tab. 6-20 Subroutine to input and pack Two character

IN2,	HEX	?	/ Save return address
FST,	SKI		/ Check if FGI= 1 ?
	BUN	FST	/ Loop (FGI = 0)
	INP		/ Input first character
	~~OUT~~		/ Echo back
	BSA	SH4	/ Shift left 4 bit
	BSA	SH4	/ Again(total 8 bit shift)
SCD,	SKI		
	BUN	SCD	
	INP		/ Input second character
	~~OUT~~		/ Echo back
	BUN	IN2 I	/ Return



◆ Store Input Character in Buffer

Tab. 6-21 Program to store input character in buffer

	LDA	ADS	/ Load buffer address A= 500
	STA	PTR	/ PTR= 500
LOP,	BSA	IN2	/ Get a character (Tab. 6-20)
	STA	PTR I	/ 500 character
	ISZ	PTR	/ PTR= 501
	BUN	LOP	/ Endless Loop
	HLT		
ADS,	HEX	500	/ Buffer address
PTR,	HEX	0	/ Pointer

◆ Compare Two Word

Tab. 6-22 Program to compare Two word

	LDA	WD1	/ Load first word A= WD1
	CMA		/ Make 2's complement
	INC		
	ADD	WD2	/ WD2 − WD1
	SZA		/ Skip if A=0 (Equal)
	BUN	UEQ	/ Unequal
	BUN	EQL	/ Equal
WD1,	HEX	?	/ first word
WD2,	HEX	?	/ second wor

Useful for a search procedure e.g. in look-up tables

Remarks

- Can write SH8 instead of double call to SH4
- Program uses a pointer to keep track of current empty location in the buffer.
- No counter is used in the program
- Characters are read
  – as long as they are available or
  – until the buffer reaches location 0 (after location FFFF).
  – In a practical situation - limit the size of the buffer, use a counter

Program to Service an Interrupt

- In former I/O example – **busy waiting**
  - Most running is wasted waiting for external devices to set flags
- Solved by **interrupt facility**
  - notify the computer when a flag is set.
- **Advantage**:
  - **information transfer** only **upon request** from external device.
  - **Meanwhile**, the computer performs other tasks.
- To be **effective**: other program(s) must reside in memory
  - **Multiprogramming environment**

24

Program to Service an Interrupt – cont.

- **Only one** program can be **executed** at any **given time**
  - However, **two or more** programs **may reside** in memory.
- Program currently being executed - ***running program***.
  - **Other programs** are usually **waiting** for **I/O** data.
- Interrupt facility service procedure
  - **Take care** of the **data transfer** of one (or more) program while another program is currently being executed.
  - The running program must include an **ION** instruction
    - to **turn** the interrupt **on**.
  - **When interrupt** facility is **not used**, program must include an **IOF**

Program to Service an Interrupt – cont.

- Interrupt facility - **allows** the **running program** to **proceed until** the **I/O devices set** their ready flags.
- Whenever a flag is set to 1
  - computer **completes** execution of current **instruction**
  - **Acknowledges** the interrupt.
    - The **return address** is **stored** in location **0.**
    - **Instruction** in **location 1** is **performed** (**initiates** a **service routine** for the input or output transfer)
- **Service routine** can be **stored anywhere** in memory
  - provided a **branch to the start** of the routine is stored **in location 1.**

Program to Service an Interrupt – cont.

- The service routine must have instructions to perform the following tasks:
  1. **Save contents** of processor registers.
  2. **Check** which **I/O flags** are set.
  3. **Service** the **device** whose flag is set.
  4. **Restore** contents of processor **registers**.
  5. **Turn** the **interrupt** facility **on**.
  6. **Return** to the **running program**.
– Also known as a ***Context switching***

Program to Service an Interrupt – cont.

- **Contents** of registers must be **the same**
  - **before** the **interrupt** and **after** the **return** to the running program
  - **otherwise**, the running program may be in **error**
- Service routine may use these **registers**
  - necessary to **save** their **contents** at the beginning of the routine
  - **Restore** them at the **end**.
- Device priority – according to **checking order** of flags
  - higher priority is serviced first, lower served afterwards
- Devices are **serviced one at a time**
  - Although **two or more** flags **may be set** at the **same time**
- During an interrupt other interrupts are ignored
  - Service routine **must turn the interrupt on before returning** to the running program (enable further interrupts)
  - **The interrupt facility should not be turned on until after the return address is inserted into the program counter.**

Program to Service an Interrupt

TABLE 6-23 Program to Service an Interrupt

Location				
0	ZRO,	—	/Return address stored here	
1		BUN SRV	/Branch to service routine	
100		CLA	/Portion of running program	
101		ION	/Turn on interrupt facility	
102		LDA X		
103		ADD Y	/Interrupt occurs here	
104		STA Z	/Program returns here after interrupt	
.		.		
.		.		
.			/Interrupt service routine	
200	SRV,	STA SAC	/Store content of *AC*	
		CIR	/Move *E* into AC(15)	Store registers
		STA SE	/Store content of *E*	
		SKI	/Check input flag	
		BUN NXT	/Flag is off, check next flag	
		INP	/Flag is on, input character	Handle Input
		OUT	/Print character	
		STA PT1 I	/Store it in input buffer	
		ISZ PT1	/Increment input pointer	
	NXT,	SKO	/Check output flag	
		BUN EXT	/Flag is off, exit	
		LDA PT2 I	/Load character from output buffer	Handle Output
		OUT	/Output character	
		ISZ PT2	/Increment output pointer	
	EXT,	LDA SE	/Restore value of AC(15)	
		CIL	/Shift it to *E*	Restore registers
		LDA SAC	/Restore content of *AC*	
		ION	/Turn interrupt on	
		BUN ZRO I	/Return to running program	
	SAC,	—	/*AC* is stored here	
	SE,	—	/*E* is stored here	
	PT1,	—	/Pointer of input buffer	
	PT2,	—	/Pointer of output buffer	

Alon Schclar, Tel-Aviv College, 2009

taken from M. Mano/Computer Design and Architecture 3rd Ed.

◆ Interrupt Program

- Interrupt Condition
  - » Interrupt F/F **R** = 1
    when **IEN** = 1 ***and*** [**FGI *or* FGO** = 1]
  - » Save return address at **0000**
  - » Jump to **0001** (Interrupt Start)
- Interrupt Service Routine(ISR)
  - » 1) Save Register (**AC**, **E**)
  - » 2) Check Input or Output Flag
  - » 3) Input or Output Service Routine
  - » 4) Restore Register (**AC**, **E**)
  - » 5) Interrupt Enable (**ION**)
  - » 6) Return to the running program

Interrupt Here

Location					
		ORG	0		
0	ZRO,	HEX	?		/ Save Interrupt Return Address
1		BUN	SRV		/ Jump to ISR
		ORG	100		/ Main program
100		CLA			
101		ION			/ Turn on Interrupt(IEN= 1)
102		LDA	X		
103		ADD	Y		/ Interrupt occurs here
104		STA	Z		/ Return Address(104)
		ORG	200		
200	SRV,	STA	SAC		/ Save A to SAC
201		CIR			/ Move A into A(15)
202		STA	SE		/ Sav e
203		SKI			/ Check if FGI= 1?
204		BUN	NXT		/ No, FGI= 0, Check FGO
205		INP			/ Yes, FGI= 1, Character Input
206		OUT			/ Echo back
207		STA	PT1	I	/ Store in input buffer(PT1)
208		ISZ	PT1		/ PT1 + 1
	NXT,	SKO			/ Check if FGO= 1?
		BUN	EXT		/ No, FGO= 0, Exit
		LDA	PT2	I	/ Yes, FGO= 1, Get output character
		OUT			/ Character output
		ISZ	PT2		/ PT2 + 1
	EXT,	LDA	SE		
		CIL			/ Restore E
		LDA	SAC		/ Restore A
		ION			/ In
		BUN	ZRO	I	/ Return to running program(104)
	SAC,	HEX	?		
	SE,	HEX	?		
	PT1,	HEX	300		/ Input Buffer Address
	PT2,	HEX	400		/ Output Buffer Address

Exercises – Chapter 6
Part 2

All questions are taken from Chapter 6 in
M. Mano/Computer Design and Architecture 3rd Ed.

Alon Schclar, Tel-Aviv College, 2009

Exercise 4

- Write a program that evaluates the logic exclusive-OR (XOR) of two logic operands.

6-19 $z = x \oplus y = xy' + x'y = [(xy')' \cdot (x'y)']'$

```
      LDA  Y
      CMA
      AND  X
      CMA
      STA  TMP
      LDA  X
      CMA
      AND  Y
      CMA
      AND  TMP
      CMA
      STA  Z
      HLT
X,    —
Y,    —
Z,    —
TMP,  —
```

Exercise 5

- Write a program to subtract two double-precision numbers.

```
1.  ORG 100
2.  CLE
3.  LDA BL
4.  CMA
5.  ADD ONE
6.  STA RES
7.  CLA
8.  CIL
9.  STA LCR
10. LDA RES
11. ADD AL
12. STA CL
13. CLA
14. CIL
15. STA ADC
16. LDA BH
17. CMA
18. ADD AH
19. ADD LCR
20. ADD ADC
21. STA CH
22. HLT
23. ONE, HEX 1
24. RES, HEX 0
25. LCR, HEX 0
26. ADC, HEX 0
27. AL,  HEX 178A
28. AH,  HEX 1
29. BL,  HEX 0
30. BH,  HEX 1
31. CL,  HEX 0
32. CH,  HEX 0
33.      END
```

- Lines 2-5 – Negating BL. Using constant ONE instead of INC since INC does not update E
- Line 6 – Store increment result without E
- Lines 7-9 – store the carry from the 2’s complement increment
- Lines 10-12 – subtract the least significant bits
- Lines 13-15 – store the carry from the low part subtraction
- Lines 16-20 – subtract the most significant bits. Include the low part carry and the 2’s complement carry

Exercise 6

- Write a subroutine to complement each word in a block of data.
- In the calling program, the BSA instruction is followed by two parameters:
  - the starting address of the block
  - the number of words in the block

Subroutine

```
CMP,  HEX  0
      LDA  CMP I
      STA  PTR
      ISZ  CMP
      LDA  CMP I
      CMA
      INC
      STA  CTR
LOP,  LDA  PTR I
      CMA
      STA  PTR I
      ISZ  PTR
      ISZ  CTR
      BUN  LOP
      ISZ  CMP
      BUN  CMP I
PTR,  —
CTR,  —
```

Calling Program

```
BSA  CMP
HEX  100  /starting address
DEC  32   /number of words
```