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** $\mathbf{(150)_{16}}$ **to** $\mathbf{(1B3)_{16}}$ 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

Hardware - faster and expensive
Software - slower and cheaper

Software implementation

- 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

Algorithm
Fig. 6-3

Circular Right

- E = 1
- E = 0

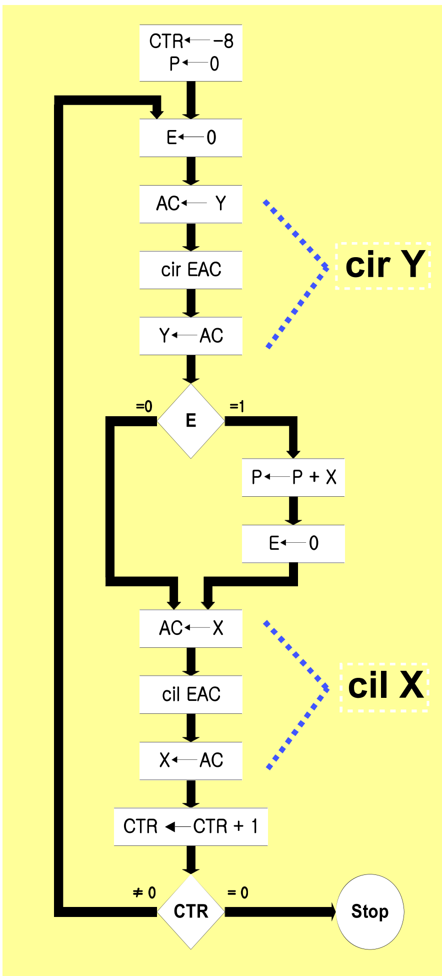
```
X = 1 1 1 1
Y = 1 0 1 1
-----------
      1 1 1 1
    1 1 1 1
  0 0 0 0
1 1 1 1
-----------
1 0 1 0 0 1 0 1
```

```
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
```

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?

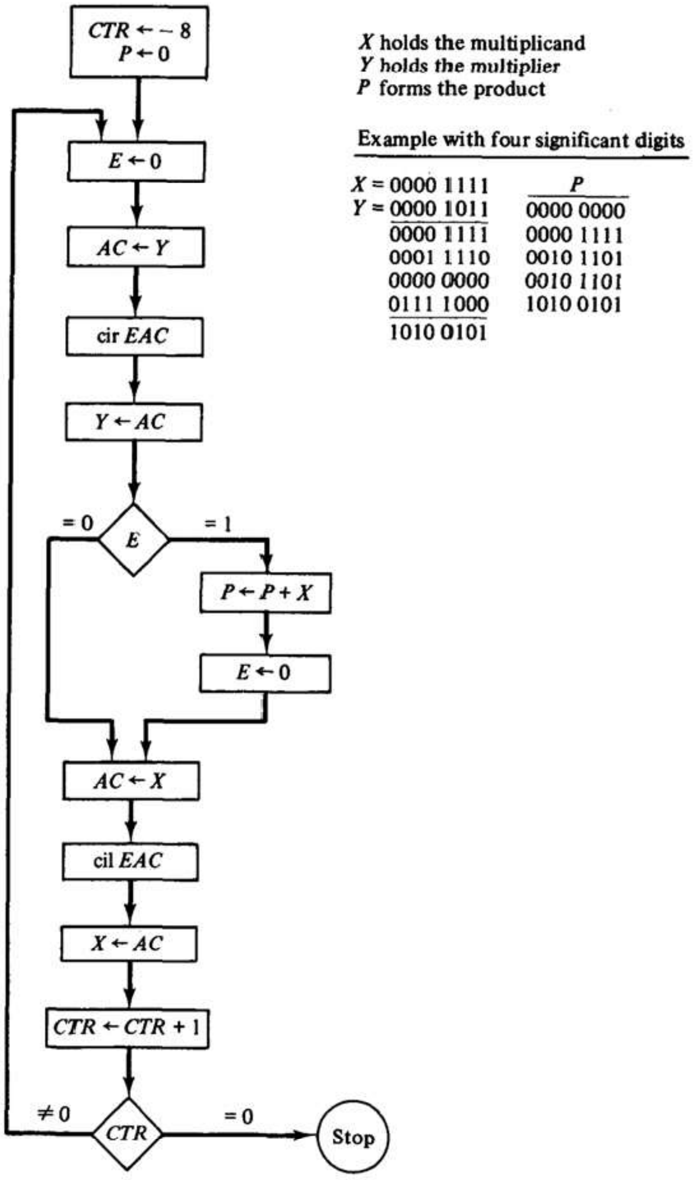


Figure 6-3 Flowchart for multiplication program.

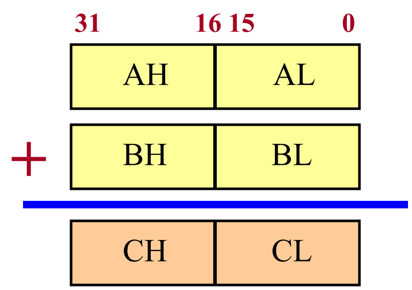
TABLE 6-14 Program to Multiply Two Positive Numbers

	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	

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 o
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,	—	

Alon Schclar, Tel-Aviv College, 2009

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

◆ Shift Operations

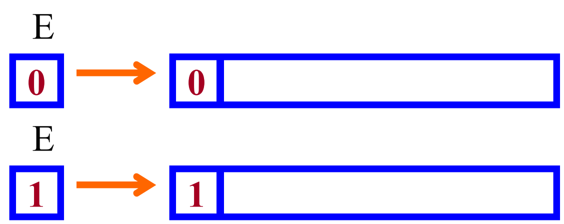
- Logical Shift : Zero must added to the extreme position
  - » Shift Right

    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

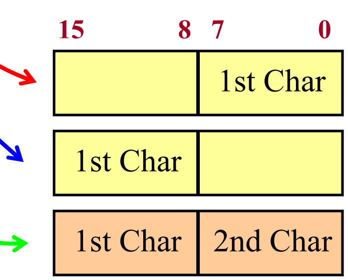
(a) Input a character			
CIF,	SKI		/ Check FGI = 1 ?
	BUN	CIF	/ Go to CIF if FGI= 0
	INP		/ Input character (FGI = 1)
	~~OUT~~		/ Echo Back
	STA	CHR	/ Store character
	HLT		
CHR,	?		/ Store character here
(b) Output a character			
	LDA	CHR	/ Load output character
COF,	SKO		/ Check FGO = 1 ?
	BUN	COF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

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)**
		STA SE	/Store content of *E*
		SKI	/Check input flag
		BUN NXT	/Flag is off, check next flag
		INP	/Flag is on, input character
		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
		OUT	/Output character
		ISZ PT2	/Increment output pointer
	EXT,	LDA SE	/Restore value of **AC(15)**
		CIL	/Shift it to *E*
		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

Store registers

Handle Input

Handle Output

Restore registers

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			/ Ine
		BUN	ZRO	I	/ Return to running program(104)
	SAC,	HEX	?		
	SE,	HEX	?		
	PT1,	HEX	300		/ Input Buffer Address
	PT2,	HEX	400		/ Output Buffer Address