

CSE 30321 Computer Architecture I


Lecture Notes 3: A Simple Computer: Simple12 And Design at Register Transfer Level

X. Sharon Hu
Department of Computer Science and Engineering

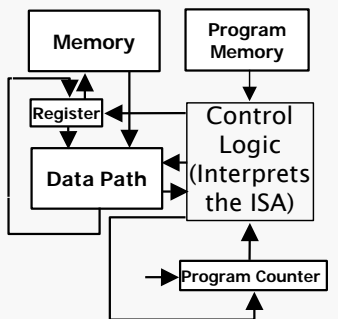
The notes are developed based on the effort of all those who have taught CSE321 before.

X.S. Hu3-1

Generic Computer Organization




- ❑ Stored Program Machine (vonNeumann Model)
- ❑ Instructions are represented as numbers
- ❑ Programs in memory are read or written just as normal data

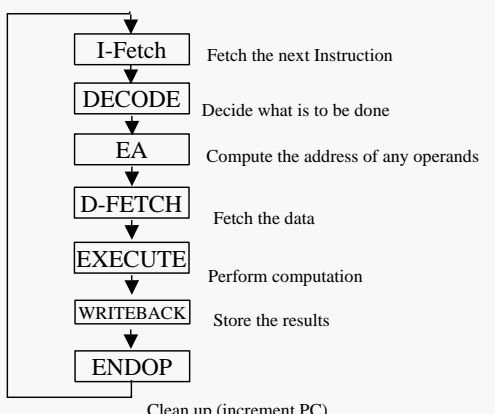


The diagram illustrates the vonNeumann Model. It shows a central 'Data Path' connected to 'Memory' and 'Program Memory'. 'Control Logic (Interprets the ISA)' is connected to the 'Data Path' and a 'Program Counter'. The 'Program Counter' provides an address to 'Program Memory'. 'Memory' is connected to the 'Data Path' and the 'Register'. The 'Register' is connected to the 'Data Path'.

X.S. Hu3-2

Typical Instruction Execution






The flowchart shows the sequence of operations for a typical instruction execution:

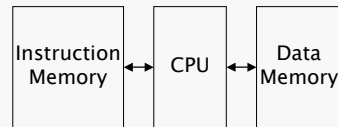
- I-Fetch**: Fetch the next Instruction
- DECODE**: Decide what is to be done
- EA**: Compute the address of any operands
- D-FETCH**: Fetch the data
- EXECUTE**: Perform computation
- WRITEBACK**: Store the results
- ENDOP**: Clean up (increment PC)

X.S. Hu3-3

Memory Organization

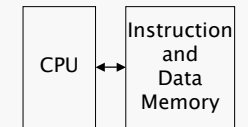


HARVARD ARCHITECTURE



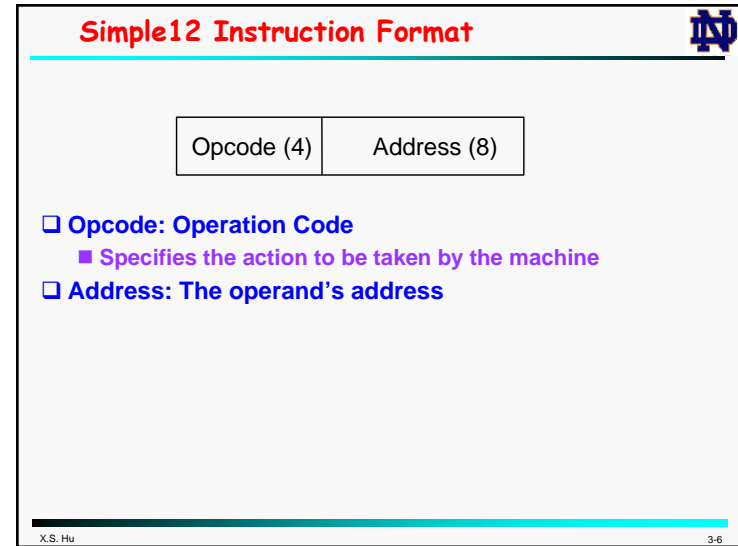
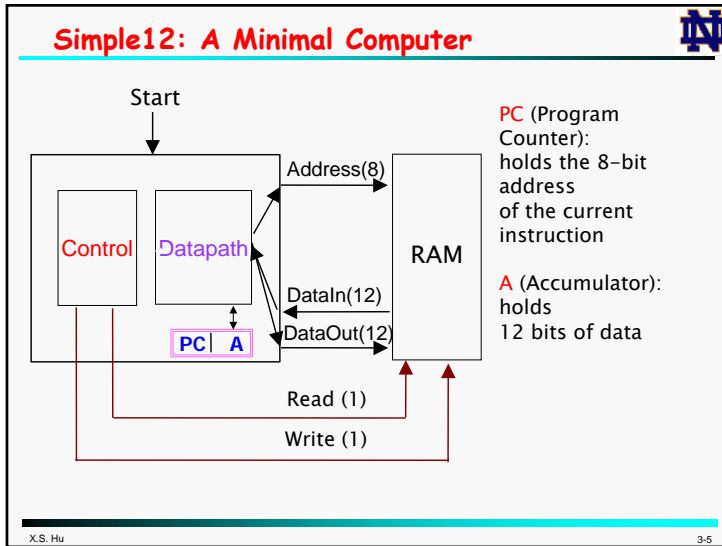
The diagram shows three boxes: 'Instruction Memory', 'CPU', and 'Data Memory'. Arrows indicate bidirectional communication between 'Instruction Memory' and 'CPU', and between 'CPU' and 'Data Memory'.

PRINCETON ARCHITECTURE



The diagram shows two boxes: 'CPU' and 'Instruction and Data Memory'. An arrow indicates bidirectional communication between them.

X.S. Hu3-4



Simple12 ISA

OPCODE	Mnemonic	RTL (What does the instruction do)
0000	JMP X	PC <- X
0001	JN X	if A < 0 then PC <- X else PC++
0010	JZ X	if A = 0 then PC <- X else PC++
0011	reserved	
0100	LOAD X	A <- M(X), PC++
0101	STORE X	M(X) <- A, PC++
0110	reserved	
0111	reserved	
1000	AND X	A <- A and M(X), PC++
1001	OR X	A <- A or M(X), PC++
1010	ADD X	A <- A + M(X), PC++
1011	SUB X	A <- A - M(X), PC++
1100	reserved	
1101	reserved	
1110	reserved	
1111	reserved	

X.S. Hu 3-7

A Simple12 Assembly Program

Problem: Given three memory locations (X, Y, and Z), use the Simple12 to find the maximum of (X, Y) and place it in Z

```

PROGRAM

0      LOAD X
1      SUB Y
2      JN B1
3      LOAD X
4      JMP SAVE
5 B1:  LOAD Y
6 SAVE: STORE Z
  
```

X.S. Hu 3-8

Program Execution (1)



PROGRAM	VALUE IN A
0 LOAD X	10
1 SUB Y	5
2 JN B1	5 (not taken)
3 LOAD X	10
4 JMP SAVE	10 (taken)
5 B1: LOAD Y	
6 SAVE: STORE Z	10
X:	10
Y:	5
Z:	n/a

X.S. Hu

3-9

Program Execution (2)



PROGRAM	VALUE IN A
0 LOAD X	
1 SUB Y	10 - 5 = 5
2 JN B1	
3 LOAD X	
4 JMP SAVE	
5 B1: LOAD Y	
6 SAVE: STORE Z	
X:	10
Y:	5
Z:	n/a

X.S. Hu

3-10

Program Execution (3)



PROGRAM	VALUE IN A
0 LOAD X	
1 SUB Y	
2 JN B1	5
3 LOAD X	
4 JMP SAVE	
5 B1: LOAD Y	
6 SAVE: STORE Z	
X:	10
Y:	5
Z:	n/a

X.S. Hu

3-11

Program Execution (4)



PROGRAM	VALUE IN A
0 LOAD X	
1 SUB Y	
2 JN B1	
3 LOAD X	
4 JMP SAVE	10
5 B1: LOAD Y	
6 SAVE: STORE Z	
X:	10
Y:	5
Z:	n/a

X.S. Hu

3-12

Program Execution (6)



PROGRAM	VALUE IN A
0 LOAD X	
1 SUB Y	
2 JN B1	
3 LOAD X	
4 JMP SAVE	
5 B1: LOAD Y	
6 SAVE: STORE Z	10
X:	10
Y:	5
Z:	10

X.S. Hu

3-13

Another Example Simple12 Program



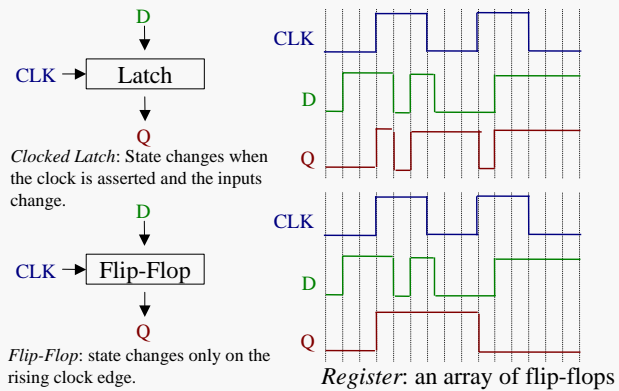
```

; A program to see if each
; item in array contains a
; particular element
; while (A[i] != 0)
;   if ( A[i] & Mask !=0)
;     A[i] = 1;
;   else
;     A[i] = 0;
;   i++;
; Data declarations!
.DATA A0 3
.DATA A1 5
.DATA A2 3
.DATA A3 8
.DATA A4 19
.DATA A5 0
.DATA Zero 0
.DATA One 1
.DATA Mask 1
L1:  LOAD  A0
     JZ    Done
     AND   Mask
     JZ    B1
     LOAD  One
     JMP   L2
B1:  LOAD  Zero
L2:  STORE A0
     LOAD  L1
     ADD   One
     STORE L1
     LOAD  L2
     ADD   One
     STORE L2
     JMP   L1
Done: .END
    
```

X.S. Hu

3-14

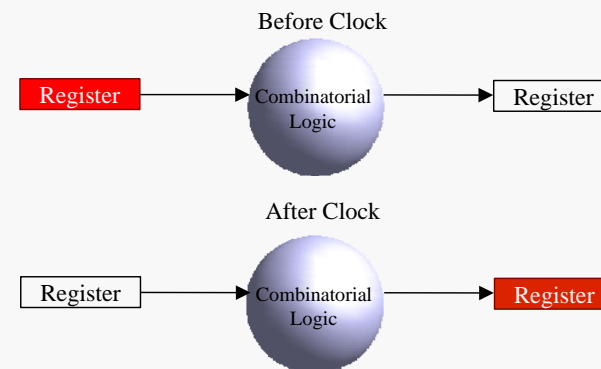
Registers



X.S. Hu

3-15

Register Transfer Operations



X.S. Hu

3-16

Register Transfer Language



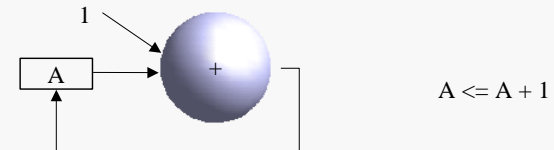
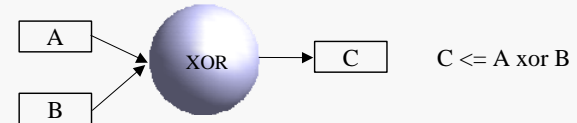
□ **Register Transfer Language (RTL):** describes the internal operation of the system in terms of a sequence of register reads, combinatorial logic, and register writes.

- Defines operations in terms of *data flow and associated control mechanisms*
- Can be relatively high level
- Forms the basis of most hardware description languages

X.S. Hu

3-17

Example Register Transfer Operations



X.S. Hu

3-18

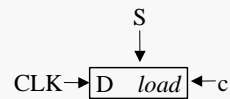
RTL Assignments



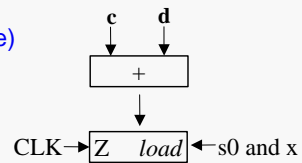
Register transfer operations are expressed as:

$$D \leq S$$

“D” (destination) gets “S” (source)



if (c) then $D \leq S$

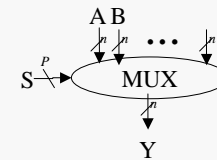


if (s0 and x) then $Z \leq c + d$

X.S. Hu

3-19

Functions and Operators



- Bit-vectors used as both operands and results
- Examples:
 - Decode(X)
 - Add(X, Y)
 - F(Q)

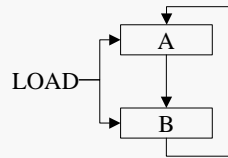
$Y \leq A$ when $s=0$ else
 $Y \leq B$ when $s=1$ else
 ...

Similar to switch/case construct in C

X.S. Hu

3-20

Parallel Assignments



$A \leftarrow B, B \leftarrow A$

Question: if $A=11$ and $B=00$ initially, what are their values after one clock cycle?

X.S. Hu

3-21

Sequencing Constructs



Each line of code is executed after the line before:

- step1: $A \leftarrow X;$
- step2: $B \leftarrow Y;$
- step3: $C \leftarrow A+B;$

Operations may occur in parallel:

- step1: $A \leftarrow X, B \leftarrow Y;$
- step2: $C \leftarrow A+B;$

Goto Statement Acceptable:

- step1: $A \leftarrow X, B \leftarrow Y;$
- step2: $C \leftarrow A + B, \text{ if } A[0] = 1 \text{ goto step1};$

X.S. Hu

3-22

RTL Notation Summarized



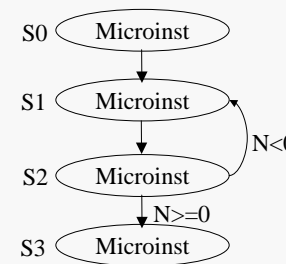
<i>SYMBOL</i>	<i>DESCRIPTION</i>	<i>EXAMPLE</i>
Names/Letters	Registers	A, B, foo
\leftarrow	Transfer into ("gets")	$A \leftarrow B$
:	indicates a control state	s0: <statement>
,	parallel microoperations	$A \leftarrow B, C \leftarrow D$
+, -	Arithmetic Operations	$A \leftarrow B+1$
&, , overline	logic operators (bitwise)	$A \leftarrow A \& B$
\ll, \gg	shift operators	$A \leftarrow B \ll 1$
if, then, else	conditional	if (c=0) then $F \leftarrow 1$ else $F \leftarrow 0$
goto	branch	goto s0

More exercise

X.S. Hu

3-23

Sequencing



- Algorithm execution is controlled by sequencing states
- Each state has an associated microinstruction
- Several parallel microoperations may occur in each state
 - implemented as register transfers
- Conditional branching

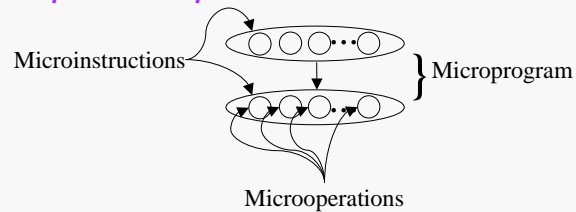
X.S. Hu

3-24

Microcoding



- AKA: Microprogramming
- Instructions are expressed as a series of *microinstructions that express each time-step of the operation.*
 - *microinstructions are composed of micro-operations*
 - *Only one microinstruction executes at a time*
 - *Each microinstruction executes its set of micro-operations in parallel*



X.S. Hu

3-25

Simple12 ISA

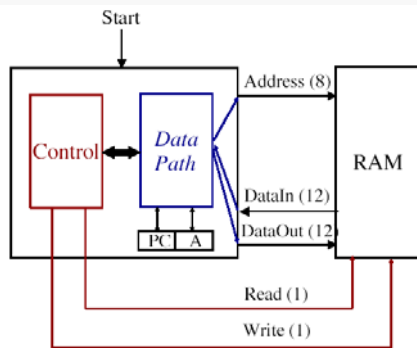


OPCODE	Mnemonic	RTL
0000	JMP X	PC <- X
0001	JN X	if A < 0 then PC <- X else PC++
0010	JZ X	if A = 0 then PC <- X else PC++
0011	reserved	
0100	LOAD X	A <- M(X), PC++
0101	STORE X	M(X) <- A, PC++
0110	reserved	
0111	reserved	
1000	AND X	A <- A and M(X), PC++
1001	OR X	A <- A or M(X), PC++
1010	ADD X	A <- A + M(X), PC++
1011	SUB X	A <- A - M(X), PC++
1100	reserved	
1101	reserved	
1110	reserved	
1111	reserved	

X.S. Hu

3-26

Simple12



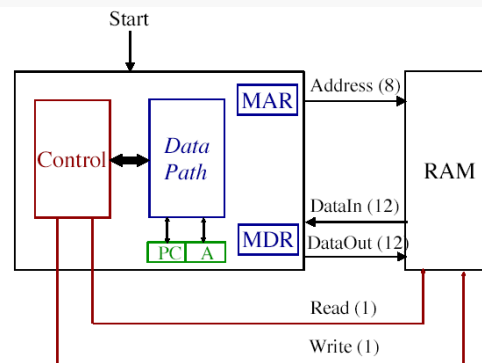
PC (Program Counter): holds the 8-bit address of the current instruction

A (Accumulator): holds 12 bits of data

X.S. Hu

3-27

Simple12: External Interfaces



VISIBLE REGISTERS
PC (Program Counter): holds the 8-bit address of the current instruction

A (Accumulator): holds 12 bits of data

INVISIBLE REGISTERS

MAR (Mem. Access Reg.): places the address for a memory access onto the address bus.

MDR (Mem. Data Reg.): transfers data to/from memory

X.S. Hu

3-28

Memory Read/Write

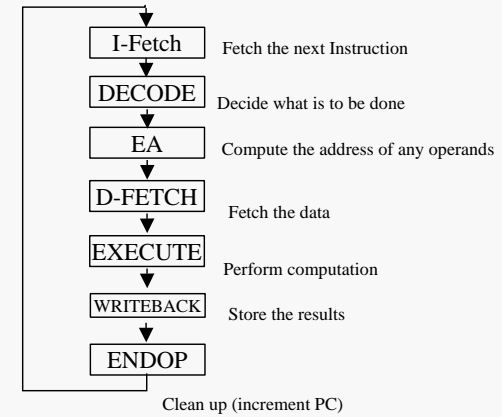


- ❑ MAR holds the address, MDR holds the value
- ❑ The control signals memory read (mem_read) and memory write (mem_write) must be set appropriately
- ❑ MEMORY READ:
 - $MAR \leftarrow address, mem_read \leftarrow 1$
- ❑ MEMORY WRITE:
 - $MAR \leftarrow address, MDR \leftarrow value, mem_write \leftarrow 1$

X.S. Hu

3-29

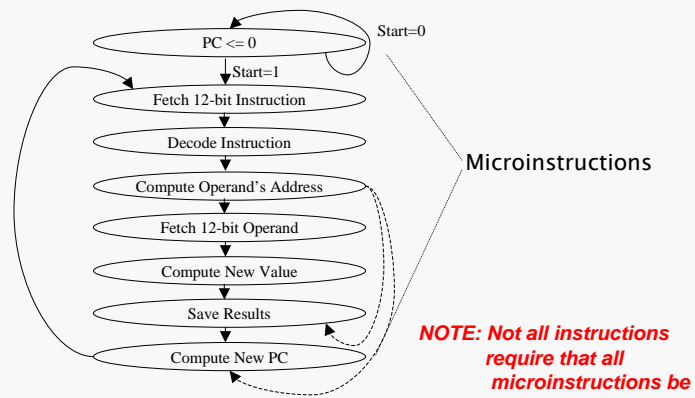
Typical Instruction Execution



X.S. Hu

3-30

Simple12 Instruction Execution



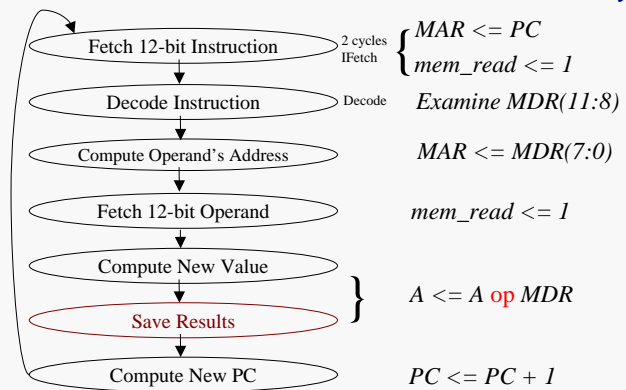
X.S. Hu

3-31

ALU Operations



op X, take the form: $A \leftarrow A \text{ op Mem}[X]$ 7 Cycles



X.S. Hu

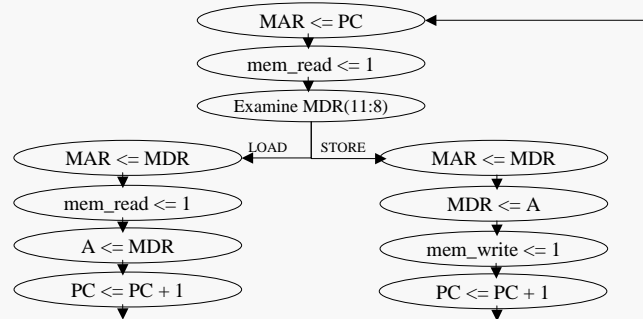
3-32

LOAD/STORE Operations



LOAD X: $A \leftarrow Mem[X]$

STORE X $Mem[X] \leftarrow A$



How many cycles?

X.S. Hu

3-33

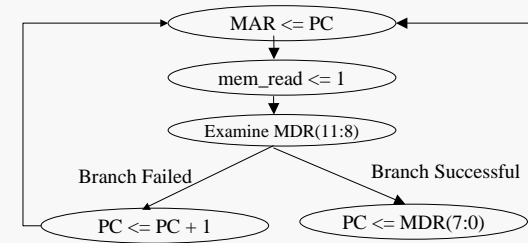
Jump Operations



JMP X
Unconditional

JN X
if $A[11]=1$

JZ X
if $A=0$



How many cycles for each instruction?

X.S. Hu

3-34

Required Datapath Operations



ALU Instructions

- AND, OR, ADD, SUB
- PC Increment

LOAD/STORE Instructions

- Register Transfers Only
- PC Increment

JUMP Instructions

- Check $A[11]$ and $A=0$
 - n_flag
 - z_flag
- Only Register Transfers

X.S. Hu

3-35

Example Program Execution



		CYCLE*	PC	A	MAR	MDR	MEM OP
EXAMPLE: Choose the max of two numbers.							
		1	00h	-	0	-	-
		2	00h	-	0	430h	Read M[0]
00h:	LOAD X (30h)	3	00h	-	0	430h	-
01h:	SUB Y (31h)	4	00h	-	30h	430h	-
02h:	JN B1 (06h)	5	00h	-	30h	007h	Read M[30h]
03h:	LOAD X (30h)	6	00h	007h	30h	007h	-
04h:	JMP SAVE (07h)	7	01h	007h	30h	007h	-
06h: B1:	LOAD Y (31h)	8	01h	007h	01h	007h	-
07h: SAVE:	STORE Z (32h)	9	01h	007h	01h	B31h	Read M[01h]
.		10	01h	007h	01h	B31h	-
.		11	01h	007h	31h	B31h	-
.		12	01h	007h	31h	00Ah	Read M[31h]
30h: X:	7	13	01h	FFDh	31h	00Ah	-
31h: Y:	10	14	02h	FFDh	31h	00Ah	-
32h: Z:	n/a						

* End of each cycle

X.S. Hu

3-36

Potential Improvements



- ❑ Cannot use opcode in the subsequent cycle as it is in MDR
 - Solution: Add an "Instruction Register" (IR)
 - Loaded from the memory bits 11-8 at the same time as MDR
- ❑ A cycle is wasted in each instruction moving PC to MAR
 - Solution: update PC and MAR simultaneously
- ❑ The only writes to memory come from A
 - Solution: tie A to DataOut and save a cycle for $MDR \leftarrow A$
- ❑ Most instructions require $PC \leftarrow PC+1$
 - Solution: use more hardware
- ❑ Many microinstructions are the same (or very similar) regardless of opcode
 - Solution: group common states together

X.S. Hu

3-37

Revised Microprogram RTL



```

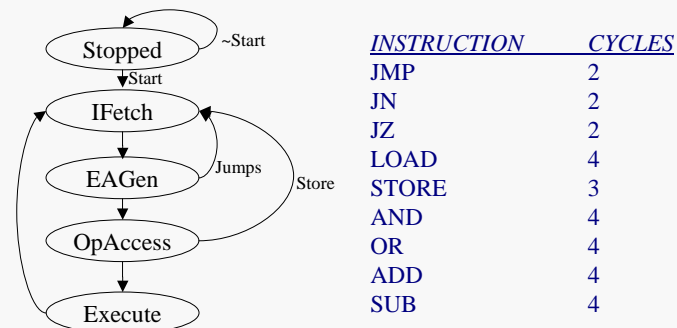
Stopped      If start=1 then (MAR <= 0, PC<=0, goto IFetch)
              else goto Stopped
IFetch:      read_mem <= 1, MDR <= DataIn, MAR<=PC+1,
              PC<=PC+1, IR<=DataIn(11:8), goto EAGen
EAGen:       if (IR=JMP) or (IR=JN and A(11)=1) or (IR=JZ and A=0)
              then MAR <= MDR(7:0), PC<=MDR(7:0) goto IFetch
              else if (IR=JN) or (IR=JZ) then goto IFetch
              else MAR <= MDR(7:0), goto OpAccess
OpAccess:    MAR <= PC,
              if IR=LOAD or IR(3:2)=10 /* i.e., an ALU operation */
              then (Read <= 1, MDR<=DataIn, goto Execute)
              else (Write <= 1, DataOut <= A, goto IFetch) /* a STORE */
Execute:     if (IR=LOAD) then A <= MDR, goto IFetch
              else if (IR=AND) then A <= A and MDR, goto IFetch
              else if (IR=OR) then A <= A or MDR, goto IFetch
              else if (IR=ADD) then A <= A + MDR, goto IFetch
              else if (IR=SUB) then A <= A - MDR, goto IFetch
    
```

NOTE: if Read and Write are not explicitly specified, they are 0

X.S. Hu

3-38

Revised State Machine



X.S. Hu

3-39

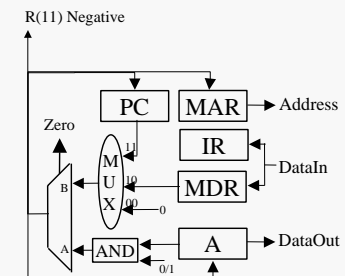
A First-Cut Datapath



Required Microoperations:

```

PC <= 0
MDR <= DataIn, IR <= DataIn,
PC <= PC + 1
MAR <= MDR(7:0),
PC <= MDR(7:0)
MAR <= PC
A <= MDR
A <= A + MDR
A <= A - MDR
A <= A and MDR
A <= A or MDR
    
```



X.S. Hu

3-40

Control Signals

Resulting Control Points:

- Load Signals** for each register
- ALU Controls** (4 bits)
 - b-invert, carry-in, op1, op0
- MUX1:** select PC, MDR, or 0
- AND:** Select 0 or A

X.S. Hu 3-41

ALU Control Definitions

<i>Binv</i>	<i>Cin</i>	<i>op1</i>	<i>op0</i>	<i>ACTION</i>
0	0	0	0	A and B
1	0	0	0	A and ~B
0	0	0	1	A or B
1	0	0	1	A or ~B
0	0	1	0	A + B
0	1	1	0	A + B + 1
1	1	1	0	A - B
1	0	1	0	A - B + 1

ALU also has a "zero" flag.

X.S. Hu 3-42

Moore and Mealy Machines

- Microinstructions will be issued to the data path using a **state machine**
- Two possibilities:
 - Moore Machine:** the outputs of the state machine depend **only on the current state**
 - Mealy Machine:** the outputs of the state machine depend **on the current state and the values of the inputs**

X.S. Hu 3-43

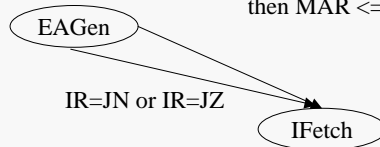
Moore and Mealy Machines

X.S. Hu 3-44

Mealy Design Problem



if (IR=JMP) or (IR=JN and A(11)=1) or
(IR=JZ and A=0)
then MAR <= PC <= MDR(7:0)



ALU needed twice in the same cycle!

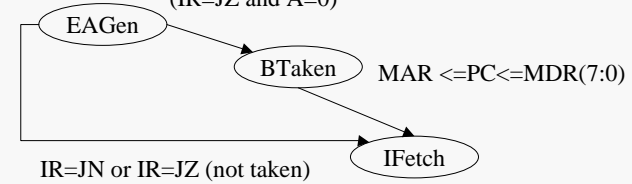
X.S. Hu

3-45

Two Solutions



First: (IR=JN and A(11)=1) or
(IR=JZ and A=0)



Second: Add More Hardware (check the Z-flag outside the ALU)

X.S. Hu

3-46

Revised Timing



<u>INSTRUCTION</u>	<u>CYCLES</u>
JMP	2
JN <i>not taken</i>	2
<i>taken</i>	3
JZ <i>not taken</i>	2
<i>taken</i>	3
LOAD	4
STORE	3
AND	4
OR	4
ADD	4
SUB	4

X.S. Hu

3-47

Simple12 Control Signal Table



Q	IR	Start	Neg	Zero	Q*	LA	LP	LM	LD	LI	ALU	bMUX	aGate	Read	Write
Stopped	n/a	0	n/a	n/a	Stopped	0	0	0	0	0	n/a	0	0	0	0
Stopped		1			IFetch	0	1	1	0	0	+	PC	0	0	0
IFetch					EAGen	0	1	1	1	1	+	MDR	0	1	0
EAGen	JMP				IFetch	0	1	1	0	0	+		0	0	0
EAGen	JN		0		IFetch	0	0	0	0	0				0	0
EAGen	JZ			0	IFetch	0	0	0	0	0				0	0
EAGen	JN		1		BTaken	0	0	0	0	0				0	0
EAGen	JZ			1	BTaken	0	0	0	0	0				0	0
EAGen	LD/ST				Opnd	0	0	1	0	0	+	MDR	0	0	0
EAGen	ALUOP				Opnd	0	0	1	0	0				0	0
Opnd	LOAD				Execute	0	0	1	1	0	+	PC	0	1	0
Opnd	ALU				Execute	0	0	1	1	0	+	PC	0	1	0
Opnd	STORE				IFetch	0	0	1	0	0	+	PC	0	0	1
Execute	LOAD				IFetch	1	0	0	0	0	+	MDR	0	0	0
Execute	AND				IFetch	1	0	0	0	0	AND	MDR	1	0	0
Execute	OR				IFetch	1	0	0	0	0	OR	MDR	1	0	0
Execute	ADD				IFetch	1	0	0	0	0	ADD	MDR	1	0	0
Execute	SUB				IFetch	1	0	0	0	0	SUB	MDR	1	0	0
BTaken					IFetch	0	1	1	0	0	+	MDR	0	0	0

X.S. Hu

3-48

Control Path Implementation



□ Hardwired

□ Microprogrammed

- A control unit with its binary control values stored as words in memory (ie, a ROM)

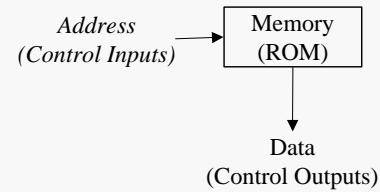
■ Big Questions:

- How do we structure the microprogram?
- How do we sequence through microinstructions?
- What are the fields?

X.S. Hu

3-49

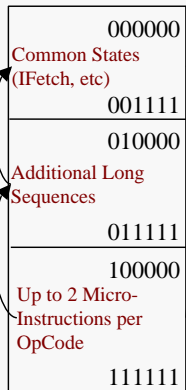
Microprogrammed Control Unit



X.S. Hu

3-50

Microprogram Structure

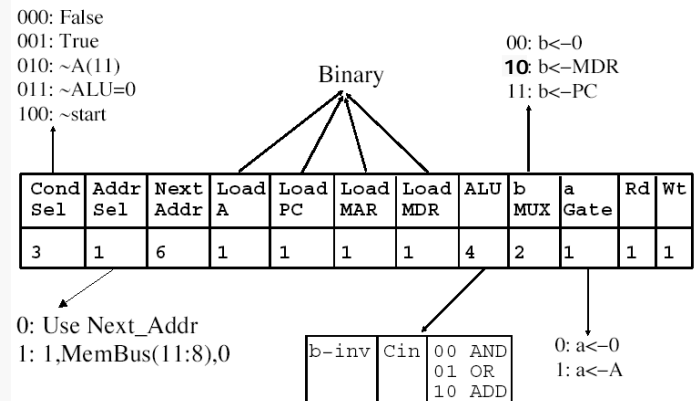


- Shared States
 - Stopped, IFetch
- 16 Possible OpCodes
 - OperandAccess and Execute sequences somewhat different for each
 - Minimum 2 non-Shared Microinstructions to be allocated
- Additional space available in table for growth

X.S. Hu

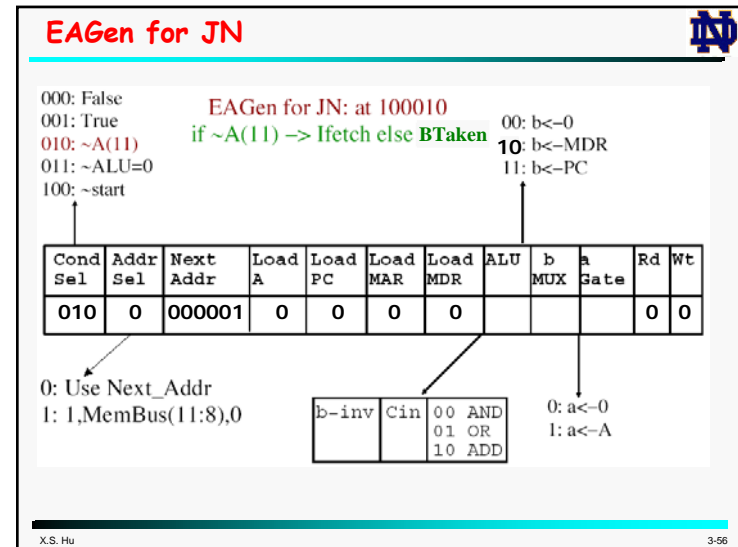
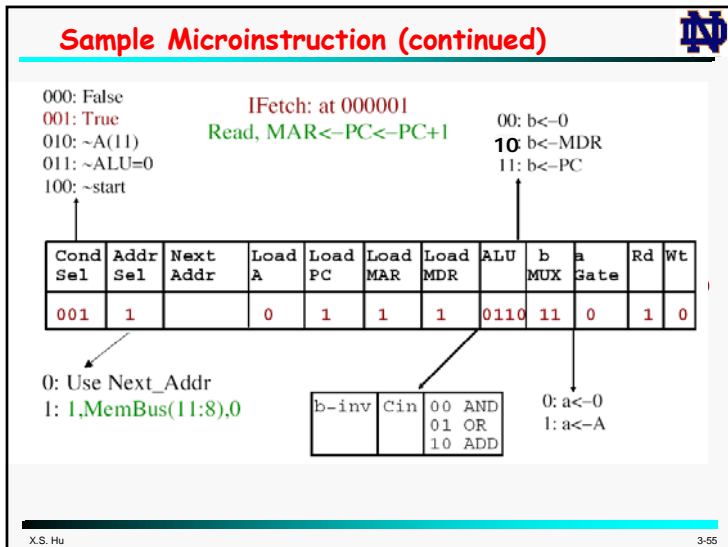
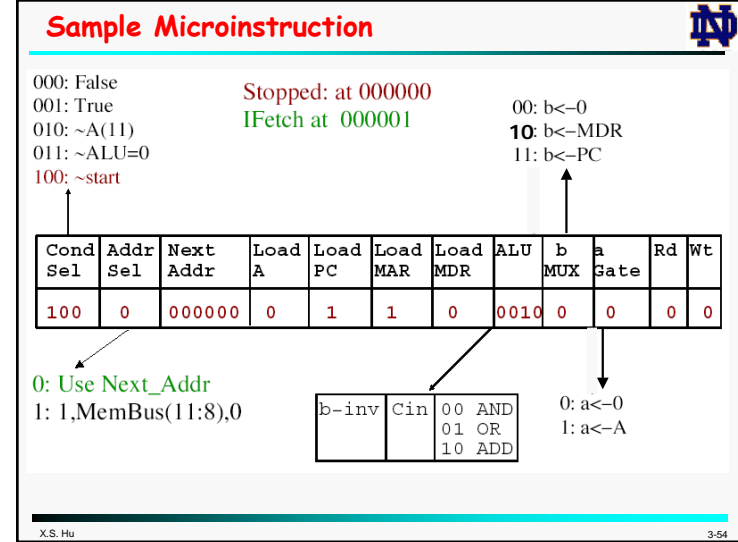
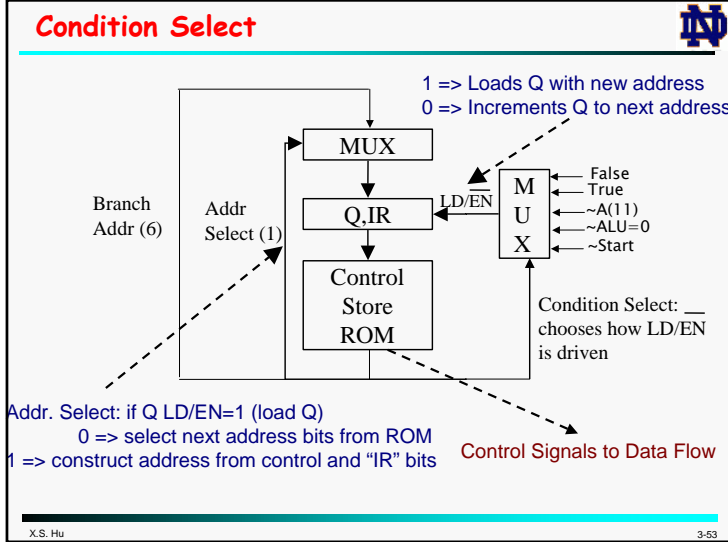
3-51

Microinstruction Template



X.S. Hu

3-52



BTaken for JN

000: False
 001: True
 010: $\sim A(11)$
 011: $\sim ALU=0$
 100: $\sim start$

BTaken for JN: at 100011
 MAR $\leftarrow PC \leftarrow MDR(7:0)$, goto IFetch

00: $b \leftarrow 0$
 10: $b \leftarrow MDR$
 11: $b \leftarrow PC$

Cond Sel	Addr Sel	Next Addr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Rd	Wt
001	0	000001	0	1	1	0	0010	10	0	0	0

0: Use Next_Addr
 1: 1, MemBus(11:8), 0

b-inv	Cin	00 AND	01 OR	10 ADD

0: $a \leftarrow 0$
 1: $a \leftarrow A$

X.S. Hu 3-57

EAGen for ADD

000: False
 001: True
 010: $\sim A(11)$
 011: $\sim ALU=0$
 100: $\sim start$

EAGen for ADD at 110100
 MAR $\leftarrow MDR(7:0)$, goto OpAcc

00: $b \leftarrow 0$
 10: $b \leftarrow MDR$
 11: $b \leftarrow PC$

Cond Sel	Addr Sel	Next Addr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Rd	Wt
000			0	0	1	0	0010	10	0	0	0

0: Use Next_Addr
 1: 1, MemBus(11:8), 0

b-inv	Cin	00 AND	01 OR	10 ADD

0: $a \leftarrow 0$
 1: $a \leftarrow A$

X.S. Hu 3-58

OpAcc for ADD

000: False
 001: True
 010: $\sim A(11)$
 011: $\sim ALU=0$
 100: $\sim start$

OpAcc for ADD at 110101
 MDR(7:0) $\leftarrow DataIn$, goto Execute

00: $b \leftarrow 0$
 10: $b \leftarrow MDR$
 11: $b \leftarrow PC$

Cond Sel	Addr Sel	Next Addr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Rd	Wt
001	0	011010	0	0	0	1				1	0

0: Use Next_Addr
 1: 1, MemBus(11:8), 0

b-inv	Cin	00 AND	01 OR	10 ADD

0: $a \leftarrow 0$
 1: $a \leftarrow A$

Any mistake? Yes, need MAR $\leftarrow PC$
 LoadMAR = 1, ALU = 0010, bMUX = 11, aGate = 0

X.S. Hu 3-59

Execute for ADD

000: False
 001: True
 010: $\sim A(11)$
 011: $\sim ALU=0$
 100: $\sim start$

Execute for ADD at 011010
 A $\leftarrow MDR(7:0) + A$, goto IFetch

00: $b \leftarrow 0$
 10: $b \leftarrow MDR$
 11: $b \leftarrow PC$

Cond Sel	Addr Sel	Next Addr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Rd	Wt
001	0	000001	1	0	0	0	0010	10	1	0	0

0: Use Next_Addr
 1: 1, MemBus(11:8), 0

b-inv	Cin	00 AND	01 OR	10 ADD

0: $a \leftarrow 0$
 1: $a \leftarrow A$

X.S. Hu 3-60

EAGen for LOAD

000: False
 001: True
 010: $\sim A(11)$
 011: $\sim ALU=0$
 100: $\sim start$

EAGen for LOAD at 101000
 $MAR \leftarrow MDR[7:0]$, goto OpAcc

00: $b \leftarrow 0$
 10: $b \leftarrow MDR$
 11: $b \leftarrow PC$

Cond Sel	Addr Sel	Next Addr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Rd	Wt
000			0	0	1	0	0010	10	0	0	0

0: Use Next_Addr
 1: 1, MemBus(11:8), 0

b-inv	Cin	00 AND 01 OR 10 ADD
-------	-----	---------------------------

0: $a \leftarrow 0$
 1: $a \leftarrow A$

X.S. Hu 3-61

OpAcc for LOAD

000: False
 001: True
 010: $\sim A(11)$
 011: $\sim ALU=0$
 100: $\sim start$

OpAcc for LOAD at 101001
 $MDR[7:0] \leftarrow DataIn$,
 $MAR \leftarrow PC$, goto Execute

00: $b \leftarrow 0$
 10: $b \leftarrow MDR$
 11: $b \leftarrow PC$

Cond Sel	Addr Sel	Next Addr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Rd	Wt
001	0	010100	0	0	1	1	0010	11	0	1	0

0: Use Next_Addr
 1: 1, MemBus(11:8), 0

b-inv	Cin	00 AND 01 OR 10 ADD
-------	-----	---------------------------

0: $a \leftarrow 0$
 1: $a \leftarrow A$

X.S. Hu 3-62

Execute for LOAD

000: False
 001: True
 010: $\sim A(11)$
 011: $\sim ALU=0$
 100: $\sim start$

Execute for LOAD at 010100
 $A \leftarrow MDR[7:0]$, goto IFetch

00: $b \leftarrow 0$
 10: $b \leftarrow MDR$
 11: $b \leftarrow PC$

Cond Sel	Addr Sel	Next Addr	Load A	Load PC	Load MAR	Load MDR	ALU	b MUX	a Gate	Rd	Wt
001	0	000001	1	0	0	0	0010	10	0	0	0

0: Use Next_Addr
 1: 1, MemBus(11:8), 0

b-inv	Cin	00 AND 01 OR 10 ADD
-------	-----	---------------------------

0: $a \leftarrow 0$
 1: $a \leftarrow A$

X.S. Hu 3-63