Introduction to Digital Circuits and Computer Design

Notes #8

"Computer Design - Datapath Unit"

Spring Quarter 2010

Avinash Kodi

Acknowledgement: Dr. Maarten Uijt de Haag

Building the Computer

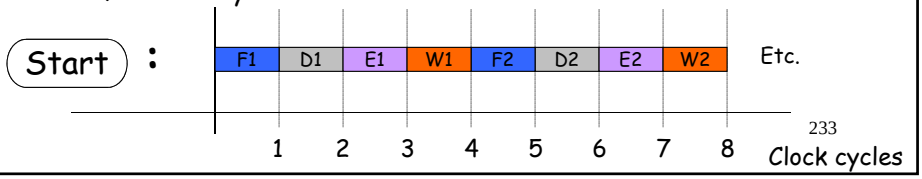
- Retrieval of instructions from memory (fetch),
- Interpretation of the instructions (decode),
- Execution of the instructions (execute),
- Reading and writing data to/from memory (read/write).

Review (Notes #1)

The instruction pipeline

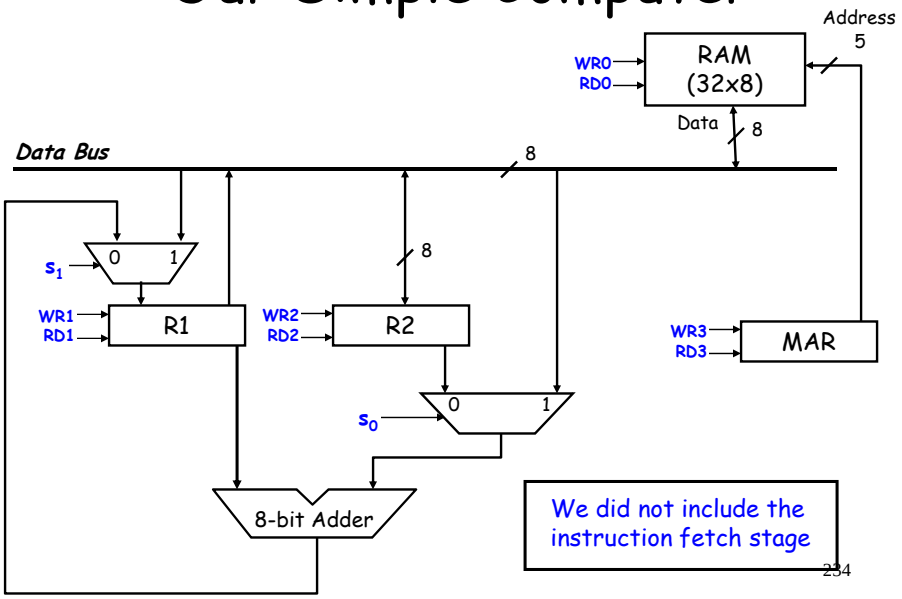
- Fetch:
  - Read an instruction from the memory
- Decode:
  - Interpret the instruction in terms of logical and arithmetic operations (micro-code)
- Execute:
  - Carry out the instruction (the micro-code)
- Write:
  - Write data back to memory or registers if necessary

F1 : Fetch cycle (instruction # 1)
D1 : Decode cycle (instruction # 1)
E1 : Execute cycle (instruction # 1)
W1 : Write cycle (instruction # 1)



233

Our Simple Computer



234

Our Simple Computer
Op-code / Operand Selection

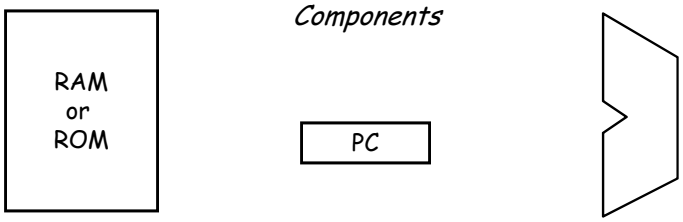
		OP-CODE	OPERAND	
Addition:	R1 ← (R1) + (R2)	000	00000	0
	R1 ← (R1) + (M)	001	XXXXX : M	1
Load/Read:	R1 ← (M)	010	XXXXX : M	1
	R2 ← (M)	100	XXXXX : M	1
Write:	M ← (R1)	101	XXXXX : M	1
	M ← (R2)	110	XXXXX : M	1

Q: What kind of instructions are these?
0-address, 1-address, 2-address, or 3-address instructions?

235

Fetching an Instruction

Remember the Program Counter (PC):
A register that contains the address of the next instruction



Instruction Storage (program storage)

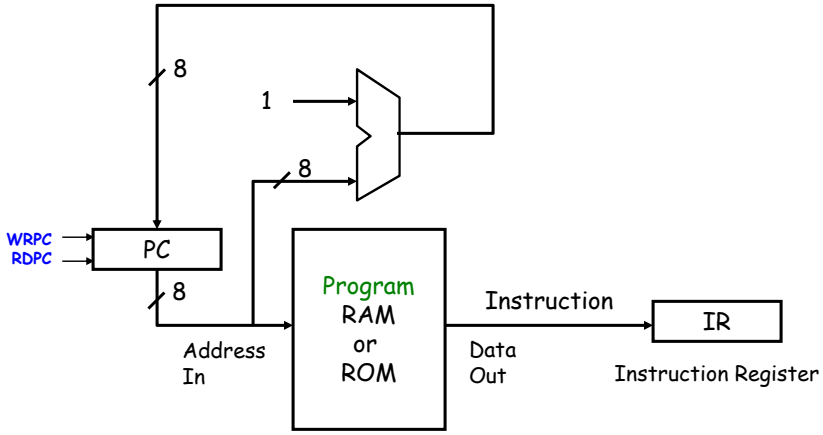
Program Counter

Adder

Why?

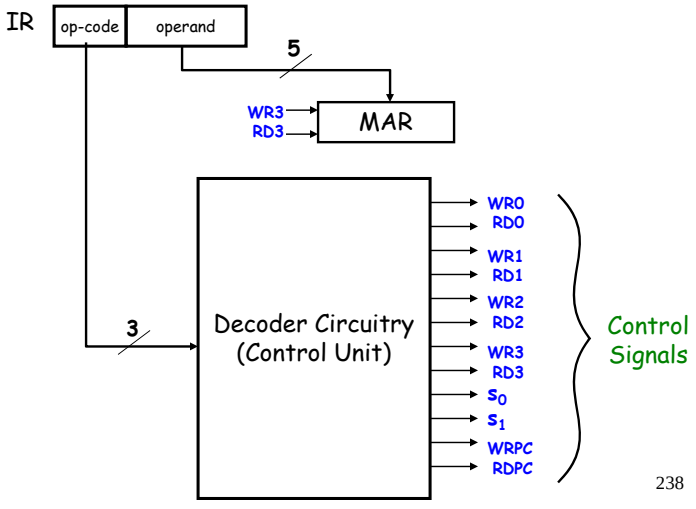
236

Fetching an Instruction



237

Decoding the Instruction



238

Decoding the Instructions

	ABC	WR0	RD0	WR1	RD1	WR2	RD2	WR3	RD3	S0	S1
R1 ← (R1) + (R2)	000	0	0	1	1	0	1	0	0	0	0
R1 ← (R1) + (M)	001	0	1	1	1	0	0	0	0	1	0
R1 ← (M)	010	0	1	1	0	0	0	0	1	X	1
Not assigned	011	0^1	0^1	0^1	0^1	0^1	0^1	0^1	0^1	0^1	0^1
R2 ← (M)	100	0	1	0	0	1	0	0	1	X	X
M ← (R1)	101	1	0	0	1	0	0	0	1	X	1
M ← (R2)	110	1	0	0	0	0	1	0	1	X	X
Not assigned	111	0^1	0^1	0^1	0^1	0^1	0^1	0^1	0^1	0^1	0^1

239

Decoding the Instructions

WR0

A \ BC	00	01	11	10
0	0	0	0	0
1	0	1	0	1

$WR0 = A\bar{B}\bar{C} + AB\bar{C} = A(B \oplus C)$

RD0

A \ BC	00	01	11	10
0	0	1	0	1
1	1	0	0	0

$RD0 = \bar{A}\bar{B}\bar{C} + \bar{A}B\bar{C} + A\bar{B}\bar{C} = \bar{A}(B \oplus C) + A\bar{B}\bar{C}$

And so on .....

$WR1 = \bar{A}\bar{B} + \bar{A}\bar{C} = \bar{A}(\bar{B} + \bar{C})$

$RD1 = \bar{A}\bar{B} + \bar{B}\bar{C} = \bar{B}(\bar{A} + \bar{C})$

$WR2 = A\bar{B}\bar{C}$

$RD2 = \bar{A}\bar{B}\bar{C} + AB\bar{C} = \bar{C}(A \odot B)$

$WR3 = 0$

$RD3 = A\bar{B} + B\bar{C}$

$s0 = \bar{B}\bar{C}$

$s1 = A\bar{B} + B\bar{C}$

should be 1 for the MAR to be filled from IR

240

Instruction Set Architecture

Let's make up some assembly mnemonics!

assembly name
to identify the instruction

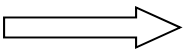
MOV	for the load and write commands	MOV	R1,[12h]
ADD	for the addition commands	ADD	R1,R2

Programming our simple computer

Example program:

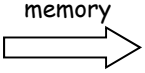
```
MOV   R1, [12h]
MOV   R2, [13h]
ADD   R1, R2
MOV   [14h], R1
```

Assembler



```
0101 0010
1001 0011
0010 0000
1011 0100
```

to program memory



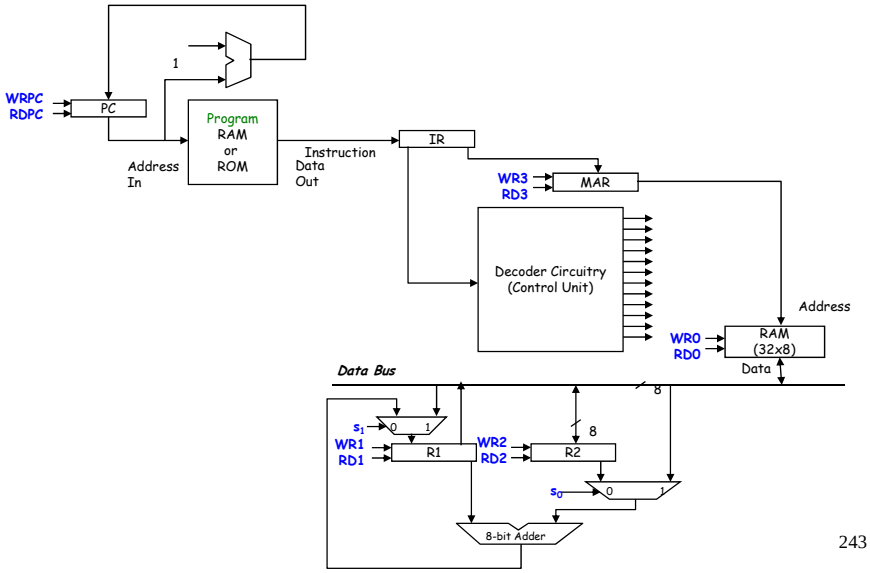
00h	0101 0010
01h	1001 0011
02h	0010 0000
03h	1011 0100

On startup
/Initially (PC) = 00h

All these are supported by our simple computer!

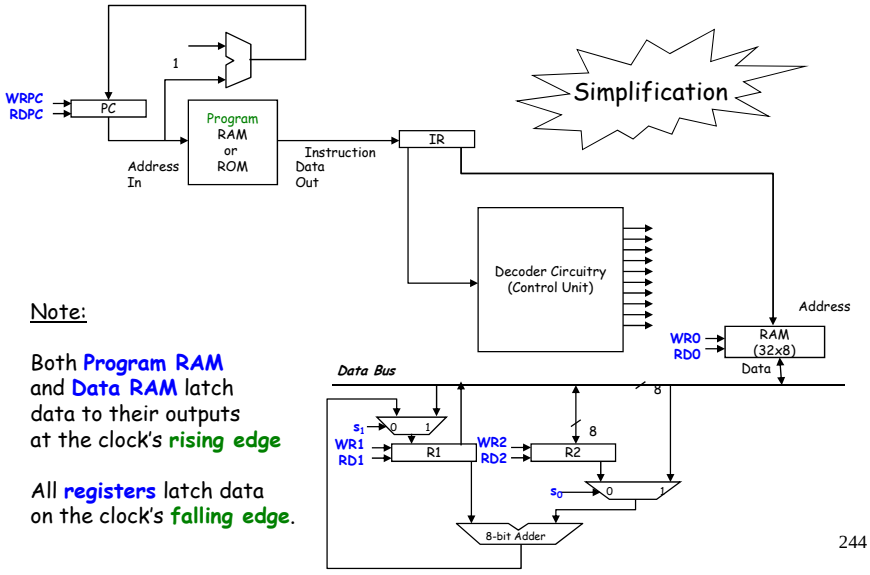
Designing the assembler is a different class (CS)

Execution of Instructions



243

Execution of Instructions



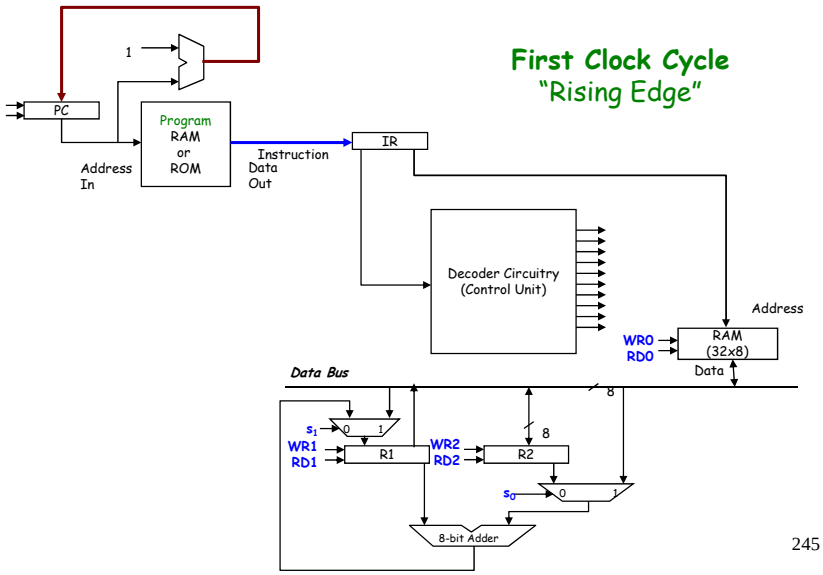
Note:

Both **Program RAM** and **Data RAM** latch data to their outputs at the clock's **rising edge**

All **registers** latch data on the clock's **falling edge**.

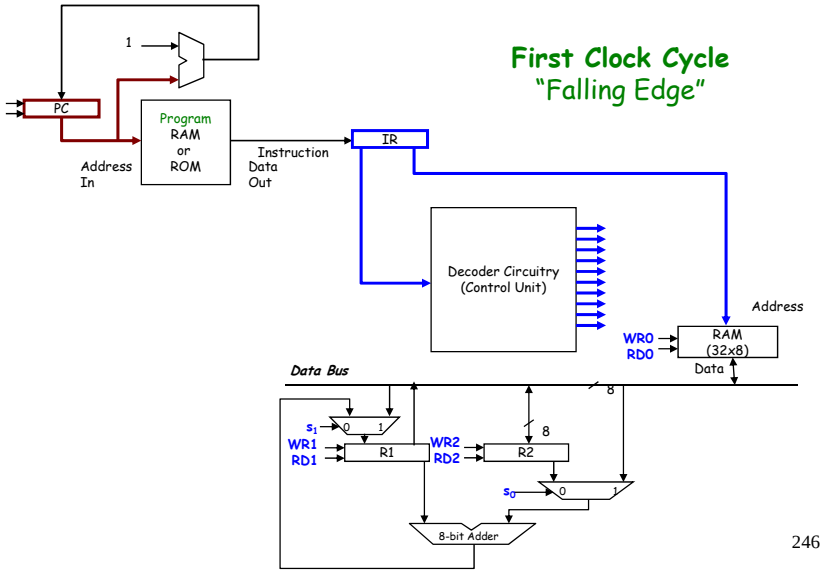
244

Execution of Instructions



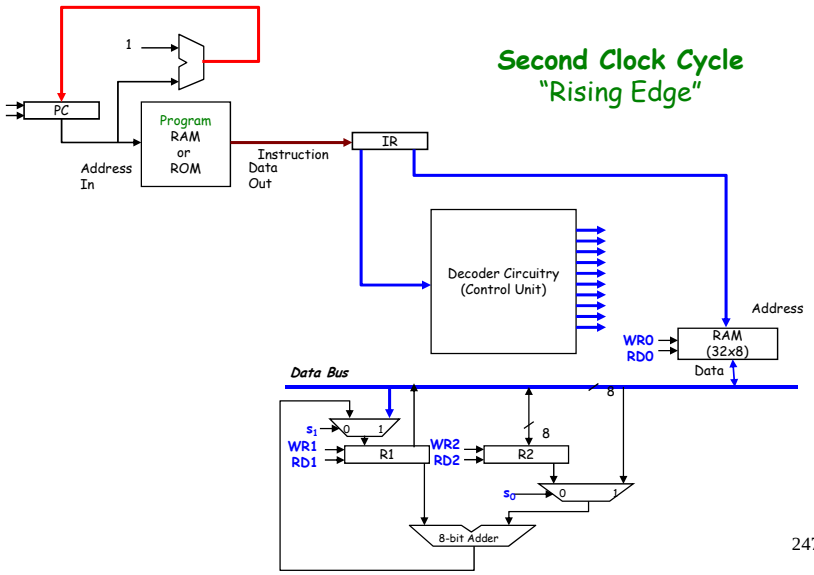
245

Execution of Instructions



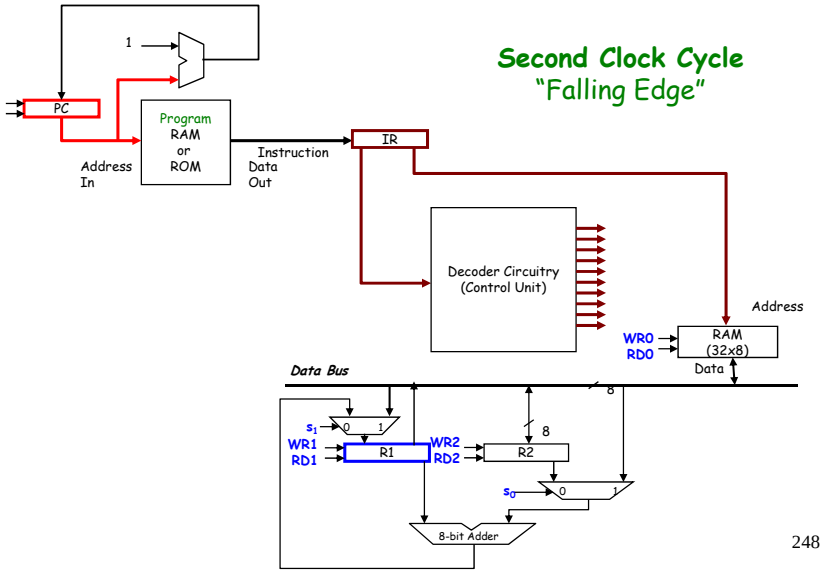
246

Execution of Instructions



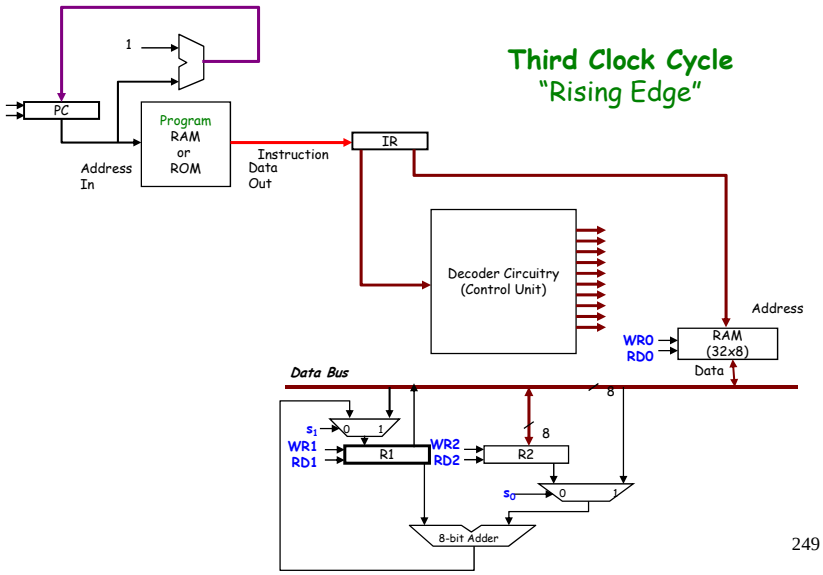
247

Execution of Instructions



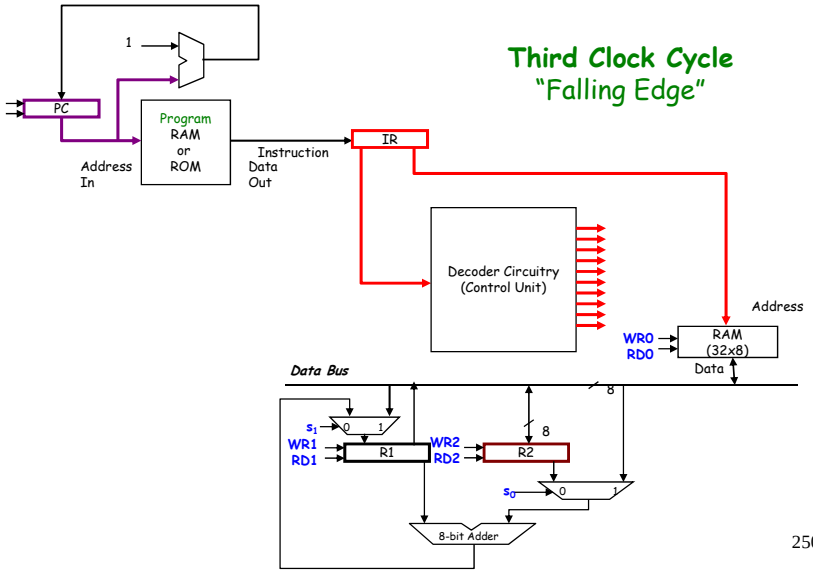
248

Execution of Instructions



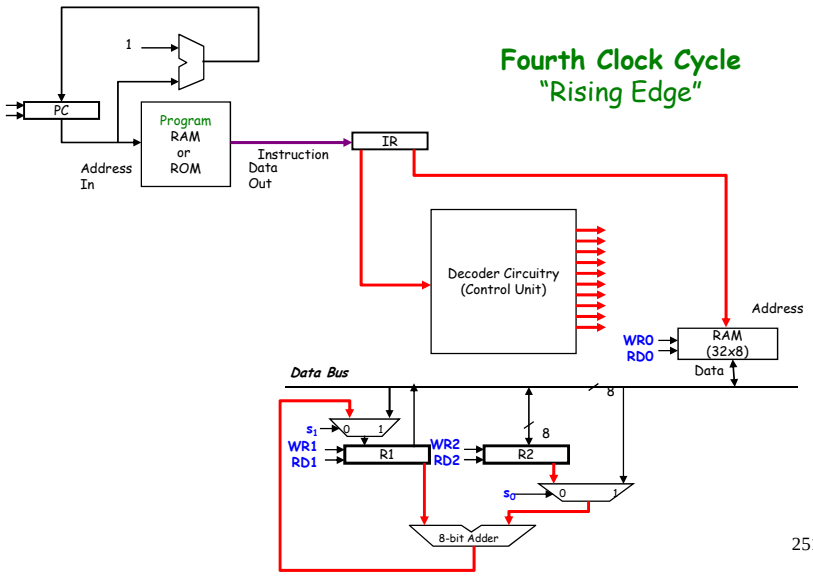
249

Execution of Instructions

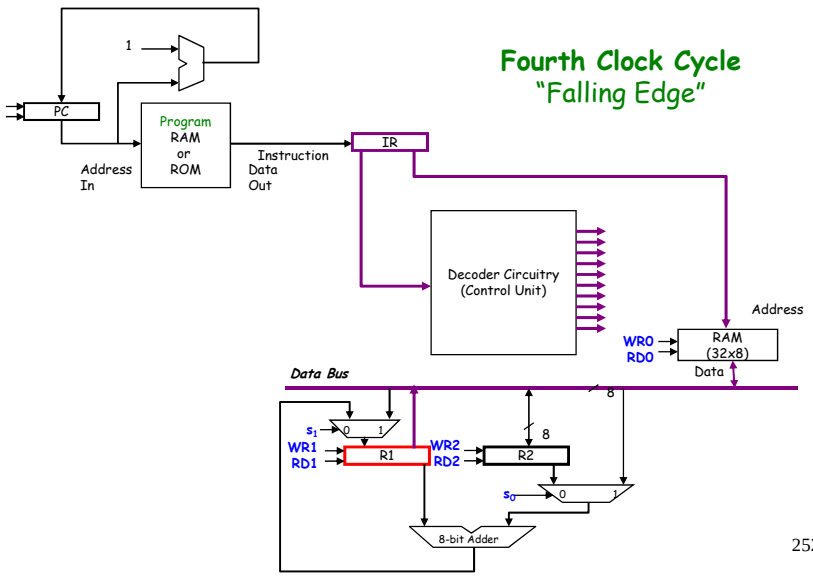


250

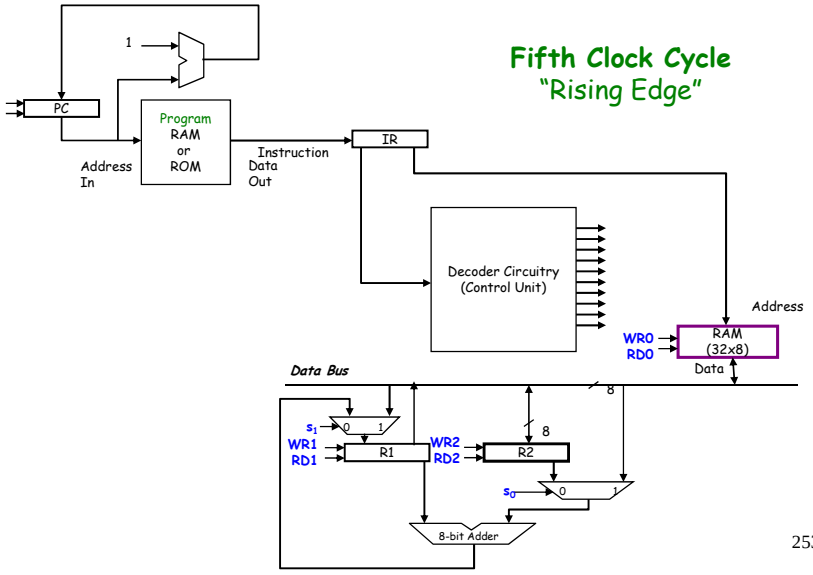
Execution of Instructions



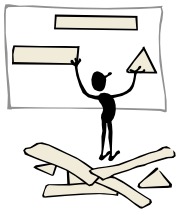
Execution of Instructions



Execution of Instructions



Datapath Design



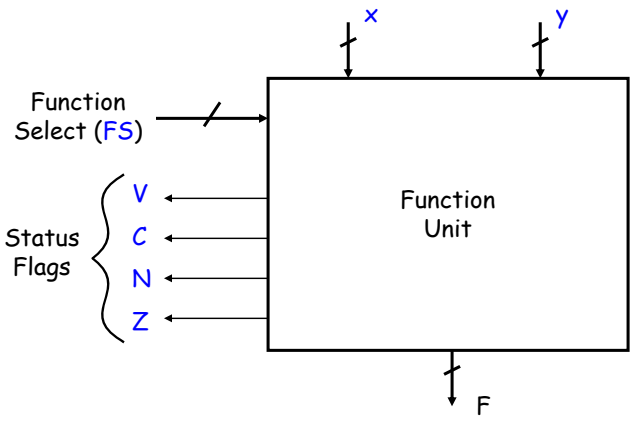
Barrel Shifters

Logic Unit

Arithmetic Unit

Register File

Function Unit
Implementation of Functions



255

Function Unit
Examples

$F = x$
$F = x + 1$
$F = x + y$
$F = x + y + 1$
$F = x + \overline{y}$
$F = x + \overline{y} + 1$
$F = sl\,x$

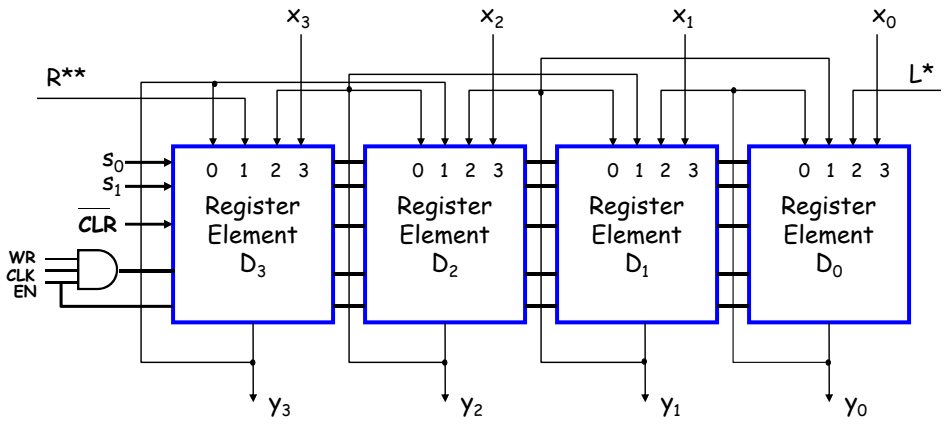
$F = x - 1$
$F = x \wedge y$ (x AND y)
$F = x \vee y$ (x OR y)
$F = \overline{x}$
$F = y$
$F = sr\,x$

FS determines what function is being executed!

256

Registers

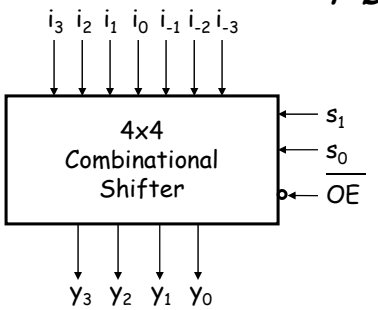
Arithmetic & Logical Shift, Rotate



257

Combinational Shifter

4-bit



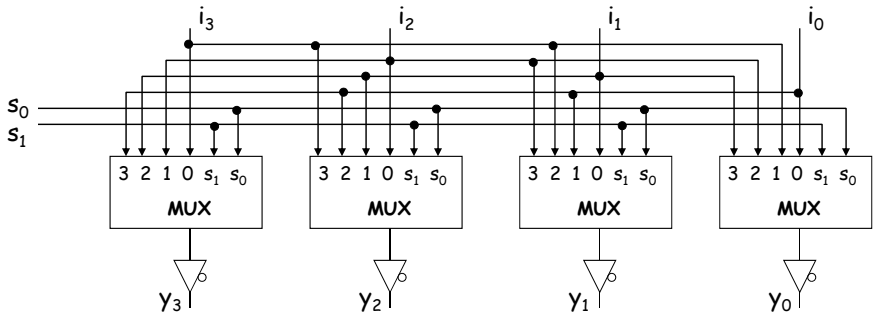
The combinational shifter can be designed using 4 multiplexers

The advantage of the combinational shifter over the register is that it only takes one clock cycle to perform any shift operation.

	Shift Count		Output				Description
$\overline{OE}$	s_1	s_0	y_3	y_2	y_1	y_0	
1	X	X	Z	Z	Z	Z	Outputs high impedance
0	0	0	i_3	i_2	i_1	i_0	No shift (pass)
0	0	1	i_2	i_1	i_0	i_{-1}	Left shift once
0	1	0	i_1	i_0	i_{-1}	i_{-2}	Left shift twice
0	1	1	i_0	i_{-1}	i_{-2}	i_{-3}	Left shift three times

258

Barrel Shifter - *4-bit*

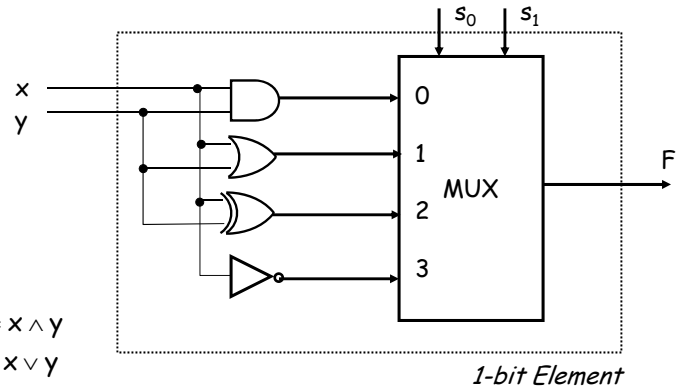


Rotate the input data bits by a number specified by the binary value of the on a set of selection lines

	Shift Count		Output				Description
$\overline{OE}$	s_1	s_0	y_3	y_2	y_1	y_0	
1	X	X	Z	Z	Z	Z	Outputs high impedance
0	0	0	i_3	i_2	i_1	i_0	No rotation (pass)
0	0	1	i_2	i_1	i_0	i_3	Rotate left once
0	1	0	i_1	i_0	i_3	i_2	Rotate left twice
0	1	1	i_0	i_3	i_2	i_1	Rotate left three times

Logic Unit

AND, OR, XOR, INV - 1-bit Element



1-bit Element

$s_1s_0 = 00 \quad : F = x \wedge y$

$s_1s_0 = 01 \quad : F = x \vee y$

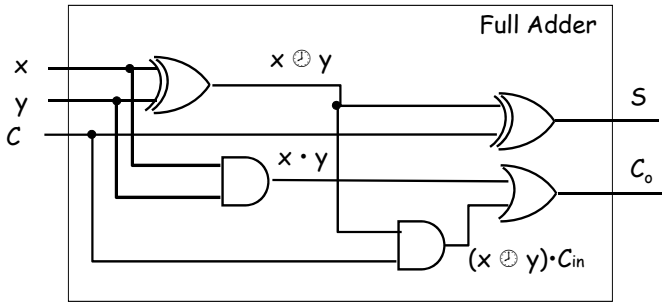
$s_1s_0 = 10 \quad : F = x \oplus y$

$s_1s_0 = 11 \quad : F = \overline{x}$

Q; How do we obtain a 4-bit or 8-bit Logic Unit?

Arithmetic Unit

Basic Element - 1-bit



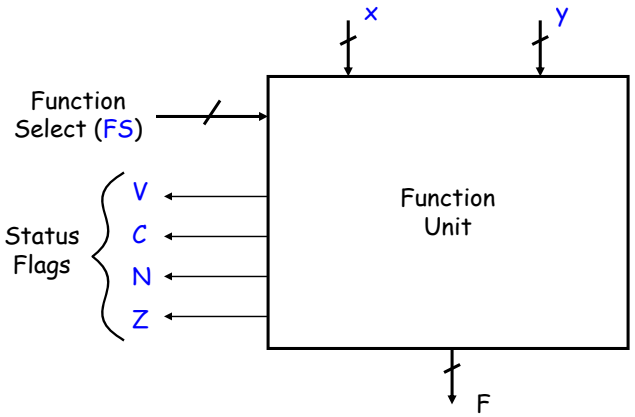
$$S = x \oplus y \oplus C = \bar{x}\bar{y}C + \bar{x}y\bar{C} + x\bar{y}\bar{C} + xyC$$

$$C_0 = (x \oplus y)C + xy = yC + xC + xy$$

261

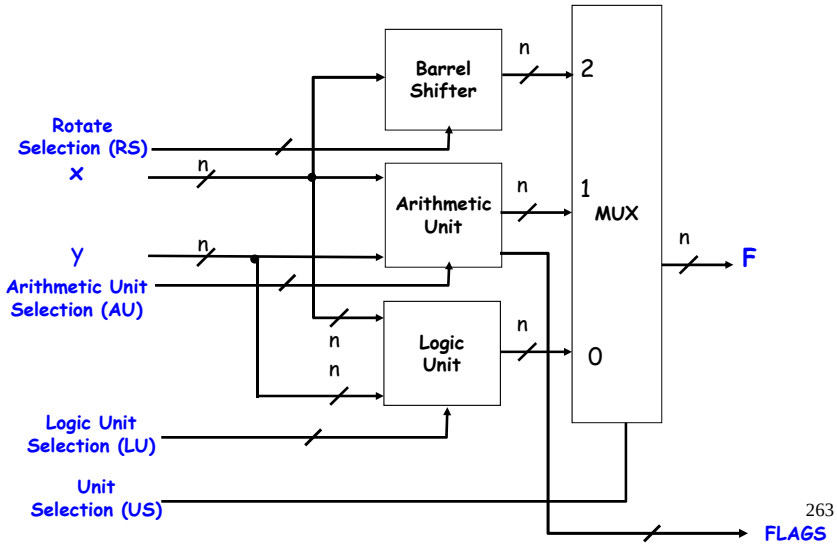
Function Unit

Implementation of Functions



262

Function Unit
Implementation



263

Function Unit
Example

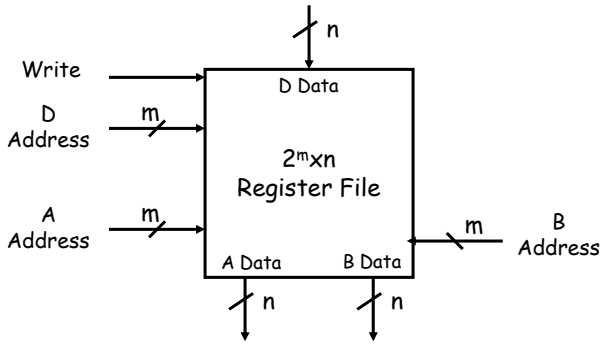
FS	RS	AU	LU	US	Micro-operation
0000 0000	00	00	00	00	F ← A ❖ B
0000 0100	00	00	01	00	F ← A • B
0000 1000	00	00	10	00	F ← A ⊕ B
0000 1100	00	00	11	00	F ← not A
0000 0001	00	00	00	01	F ← A + B
0001 0001	00	01	00	01	F ← A - B
0010 0001	00	10	00	01	F ← A + 1
0011 0001	00	11	00	01	F ← A
0000 0010	00	00	00	10	F ← sr1 A
0100 0010	01	00	00	10	F ← sr2 A
1000 0010	10	00	00	10	F ← sr3 A
1100 0010	11	00	00	10	F ← sr4 A

Q: Would you be able to make the number of control lines smaller?
Answer: Yes, there are only 12 operations; would require 4 bits;
Requires some logic to decode these bits

264

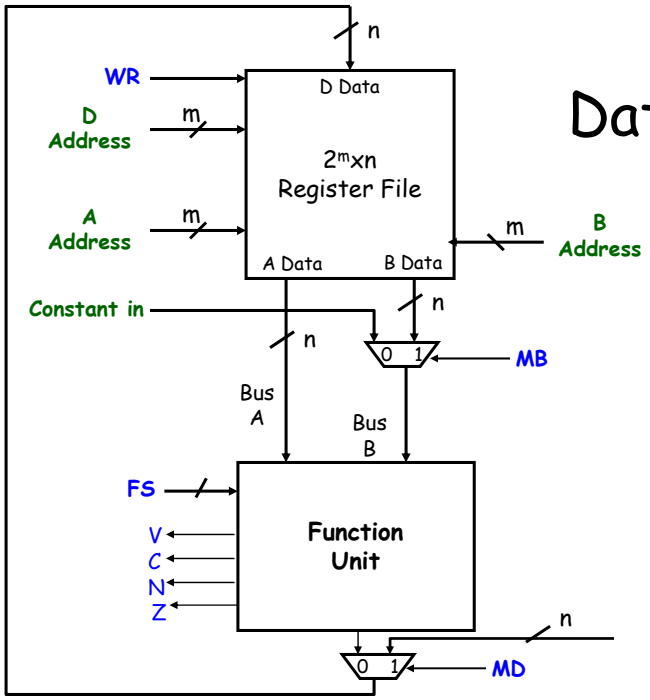
Register File

Instead of having just two registers, a typical datapath has more than four registers. Computer with 32 or even more registers are not uncommon (PowerPC)! A set of registers having common micro-operations may be organized into a register file; a fast memory that permits one or more memory locations to be read or one or more memory locations to be written to.

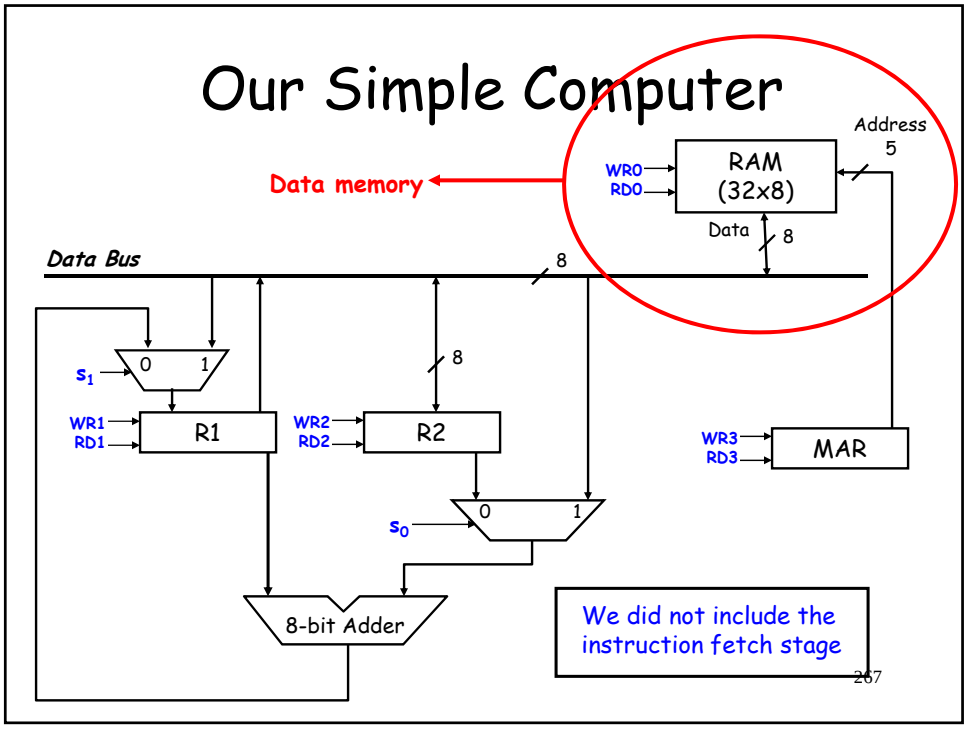


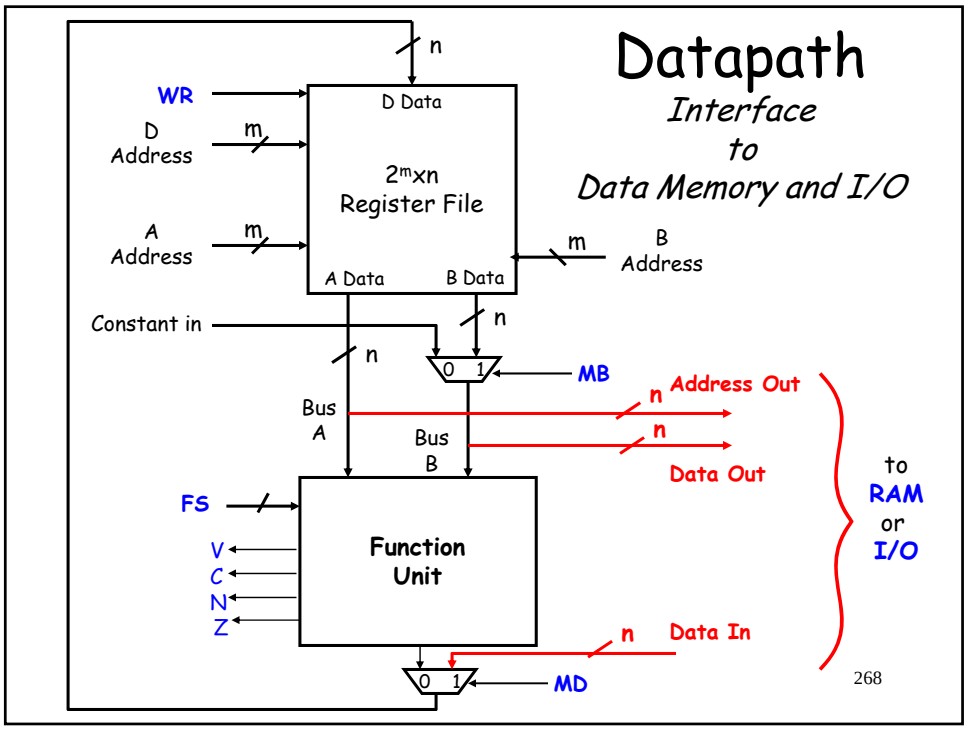
265

Datapath



266


Our Simple Computer
Address
5
Data memory
WR0
RD0
RAM
(32x8)
Data
8
Data Bus
8
s_1
0
1
8
WR1
RD1
R1
WR2
RD2
R2
WR3
RD3
MAR
s_0
0
1
8-bit Adder
We did not include the instruction fetch stage
267


Datapath
Interface
to
Data Memory and I/O
n
WR
D Data
D
Address
m
2^mxn
Register File
A
Address
m
m
B
Address
A Data
B Data
Constant in
n
n
0
1
MB
Address Out
n
Bus
A
Bus
B
n
Data Out
to
RAM
or
I/O
FS
V
C
N
Z
Function
Unit
n
Data In
0
1
MD
268