CISC260 Machine Organization and Assembly Language

Build A Simple Computer

1. Any function defined on binary input and output variables can be implemented as Boolean expression. True or False?

A. True
B. False

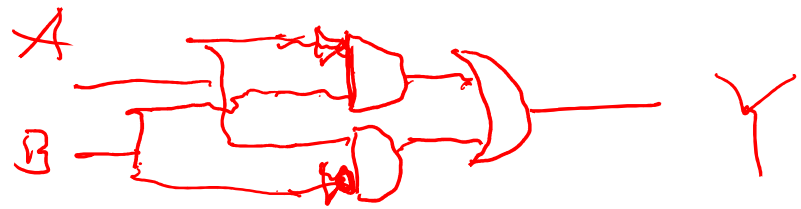
2. Which of the following is the canonical expression for XOR?

A. Y = ~ (A |B)
B. Y = ~A | ~B
C. Y= ~A & B
D. Y = (~A & B) | (A & ~B)

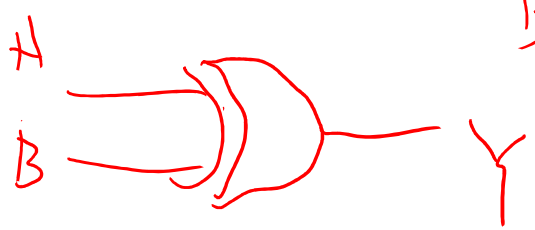
A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0

2. Which of the following is the canonical expression for XOR?

A. Y = ~ (A | B)
B. Y = ~A | ~B
C. Y= ~A & B
D. Y = (~A & B) | (A & ~B)



A	B	Y
0	0	0
0	1	1
1	0	1
1	1	0



1 Half Adder

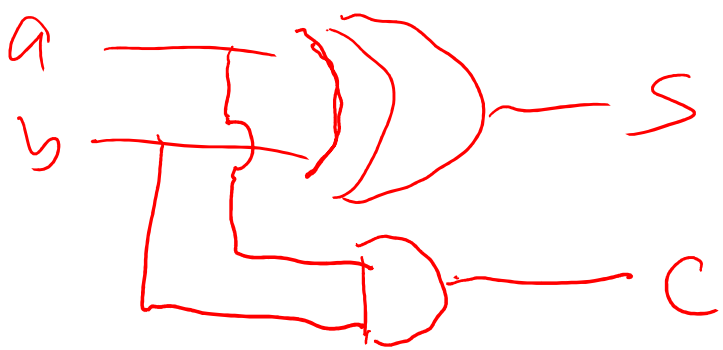
C = Carry

S = sum

$$\begin{array}{r} a \\ + \quad b \\ \hline C\ S \end{array}$$

a	b	S	C
0	0	0	0
0	1	1	0
1	0	1	0
1	1	0	1

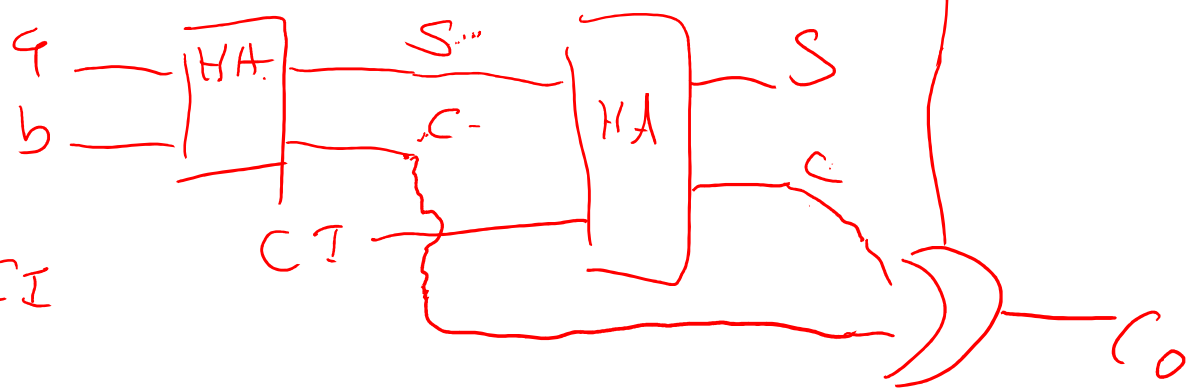
$S = f(a, b) = XOR(a, b)$

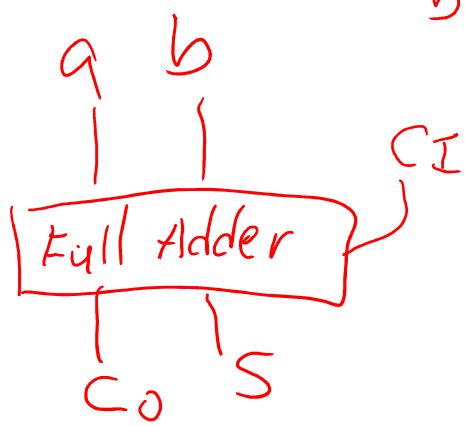


2. Full Adder

$$
\begin{array}{r}
a \\
b \\
+)\quad CI \\
\hline
Co\ S
\end{array}
$$

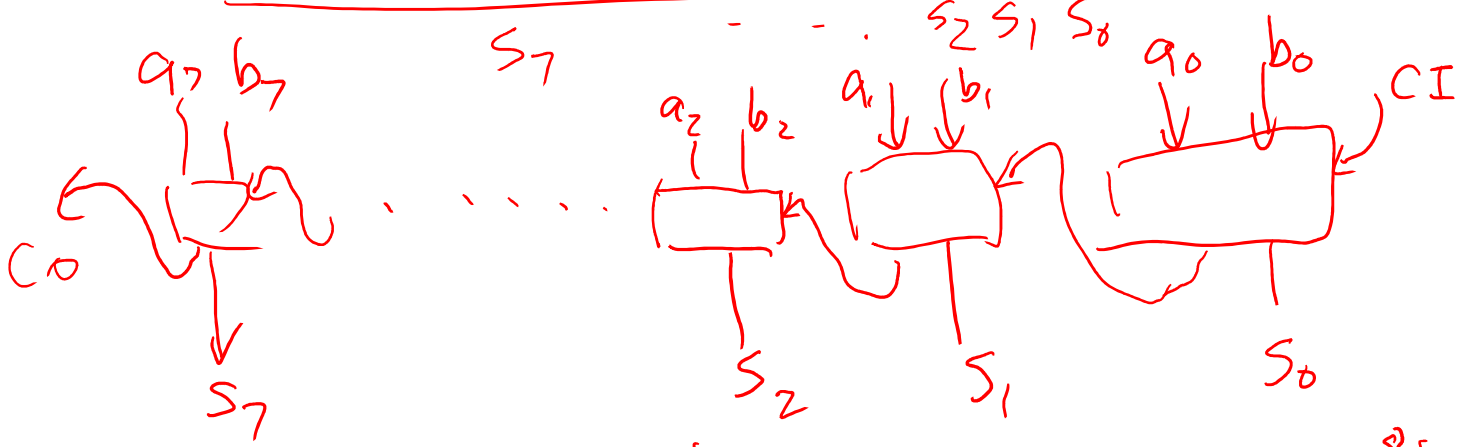
a, b, CI	S, CO
1 1 1	1 1



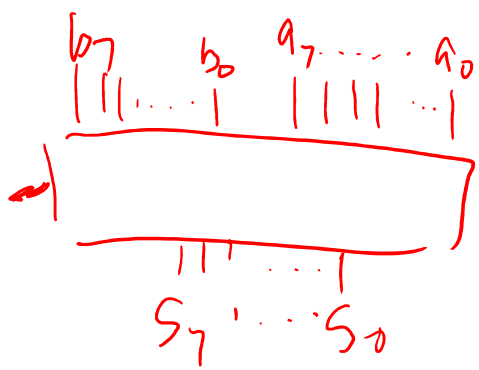


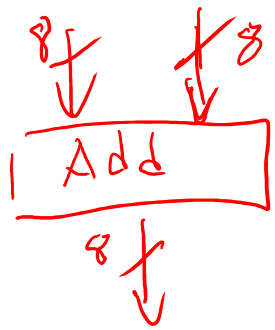
3, Adder

$$
\begin{array}{rcccccccc}
 & a_7 & a_6 & a_5 & a_4 & a_3 & a_2 & a_1 & a_0 \\
+) & b_7 & b_6 & \cdots & & & & & b_0 \\
\hline
 & s_7 & & \cdots & & & s_2 & s_1 & s_0
\end{array}
$$

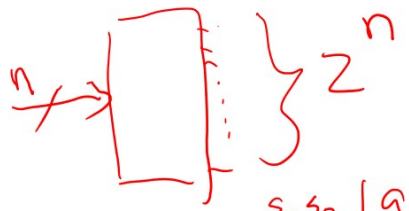


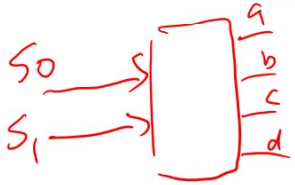
Ripple Adder



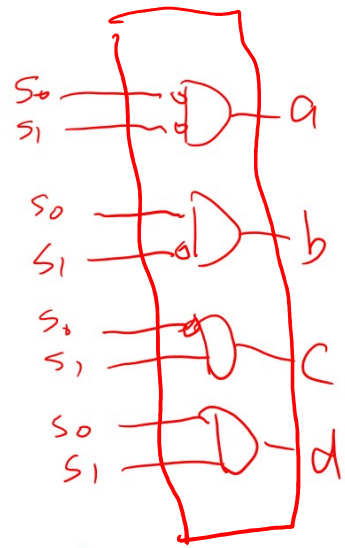


4. Decoder





S_1	S_0	a	b	c	d
0	0	1	0	0	0
0	1	0	1	0	0
1	0	0	0	1	0
1	1	0	0	0	1



Output is "one-hot" decoding of the Input

ADD
SUB
AND
NOT
OR
SLL
SRL
⋮

3. Which of the following is the canonical expression for the function defined by the following truth table?

A. Y = A.~B.~S
B. Y = A.B.~S
C. Y= ~A.B.S
D. Y = A.B.S
E. Y = A.~B.~S + A.B.~S + ~A.B.S + A.B.S

A	B	S	Y
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

3. Which of the following is the canonical expression for the function defined by the following truth table?

multiplexer

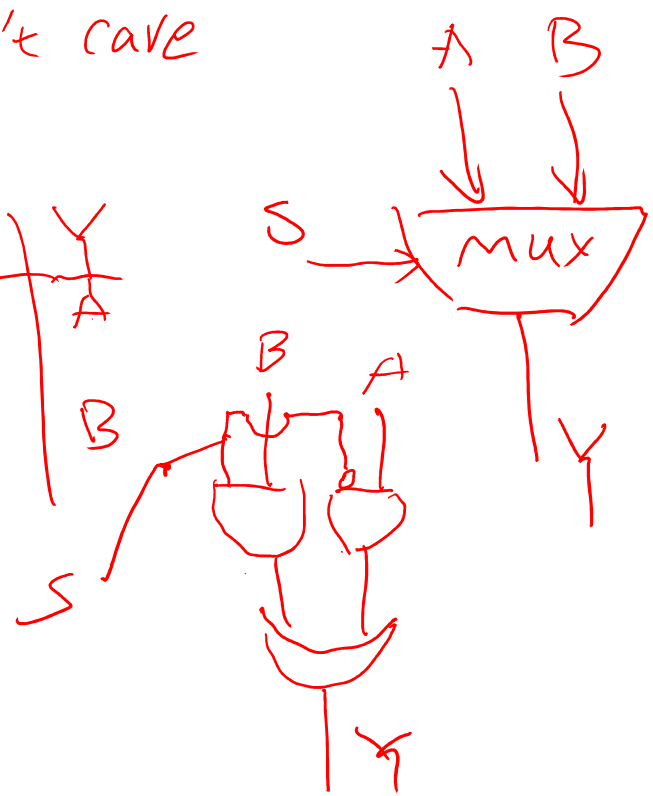
A. Y = A.~B.~S
B. Y = A.B.~S
C. Y= ~A.B.S
D. Y = A.B.S
E. Y = A.~B.~S + A.B.~S + ~A.B.S + A.B.S

X: don't care

A	B	S	Y
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

=>

A	B	S	Y
A	X	0	A
X	B	1	B



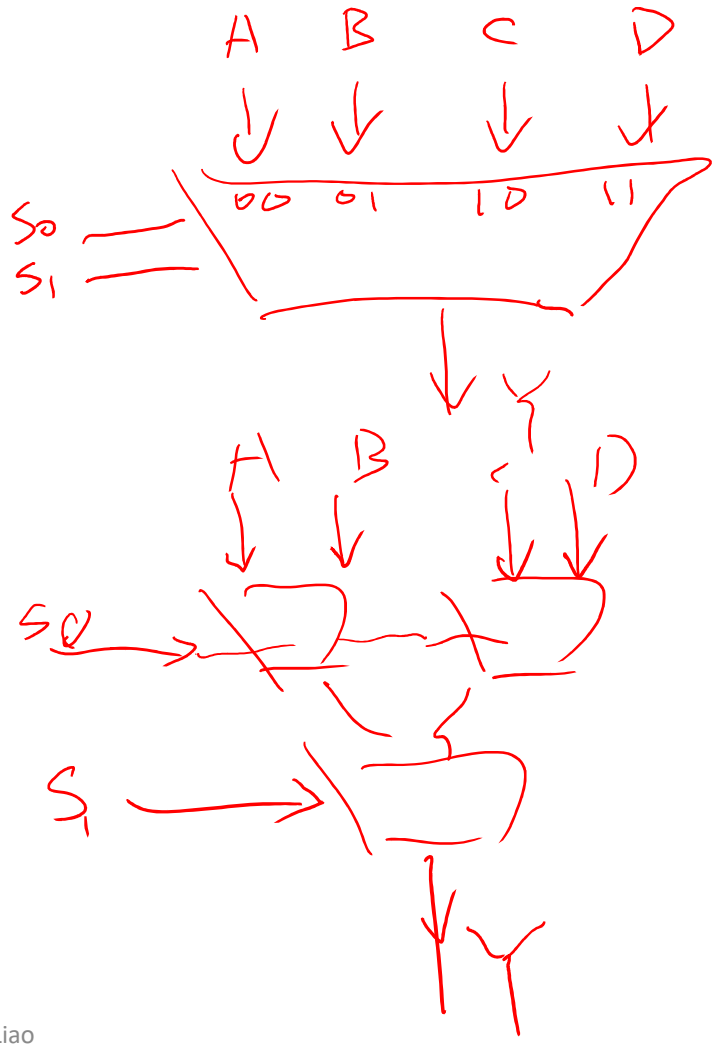
4. Are the following two Boolean expressions equivalent to each other for the function Y = f(A,B,S) defined by the following truth table?

Y = A & ~S | B & S
Y = A &~B &~S | A & B & ~S | ~A & B & S | A & B & S

A	B	S	Y
0	0	0	0
0	1	0	0
1	0	0	1
1	1	0	1
0	0	1	0
0	1	1	1
1	0	1	0
1	1	1	1

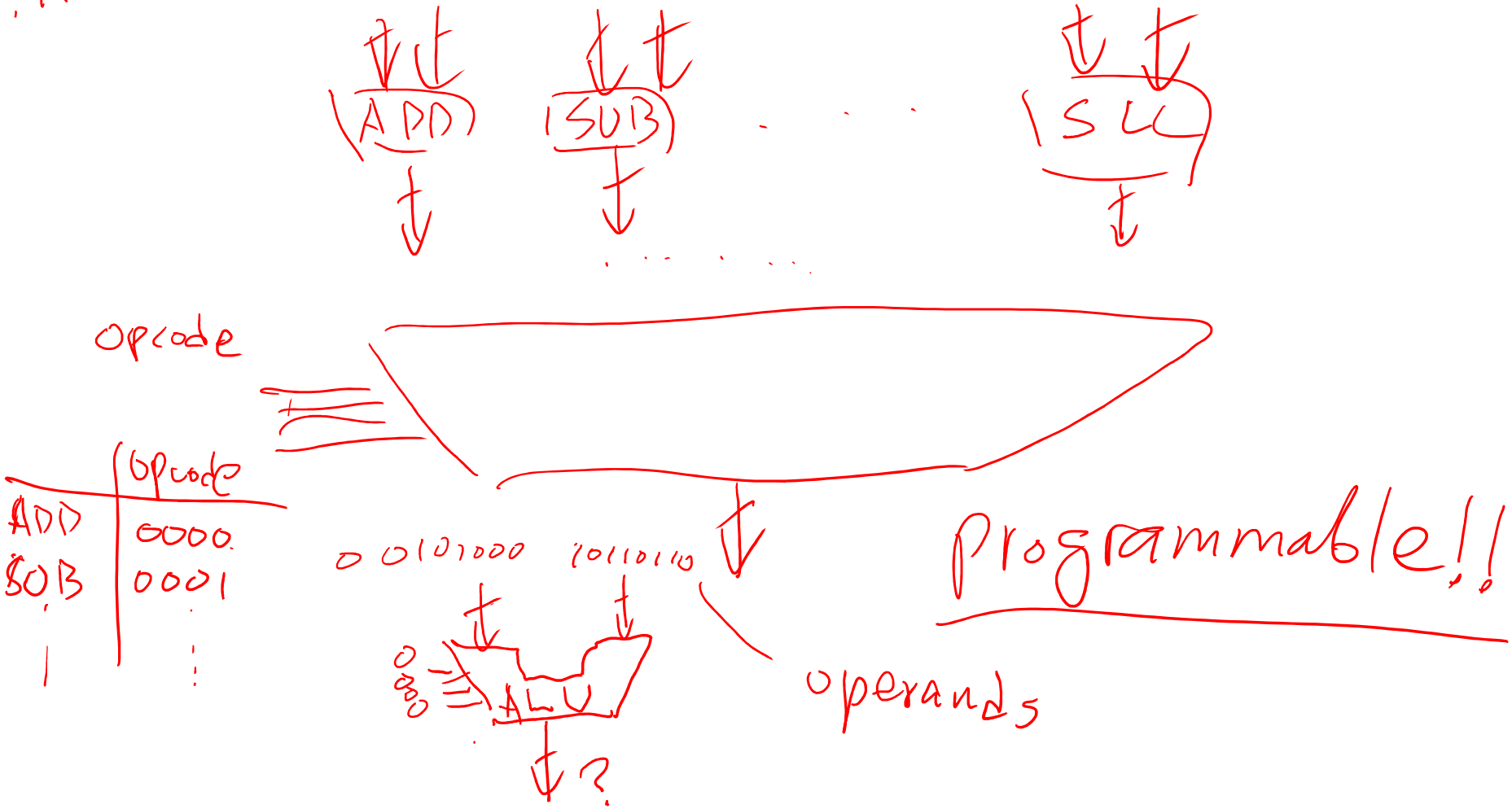
A. Yes
B. No

5 MuX



A	B	C	D	S_0	S_1	Y
A	x	x	x	0	0	A
x	B	x	x	0	1	B
x	x	C	x	1	0	C
x	x	x	D	1	1	D

6. ALU


ADD
SUB
SLL
opcode
opcode
ADD 0000
SUB 0001
0 0101000 10110110
ALU
?
operands
Programmable!!

Insanity: doing the same thing over and over again and expecting different results? -Albert Einstein

Do you agree or not?

A. Yes
B. No

7. Storage

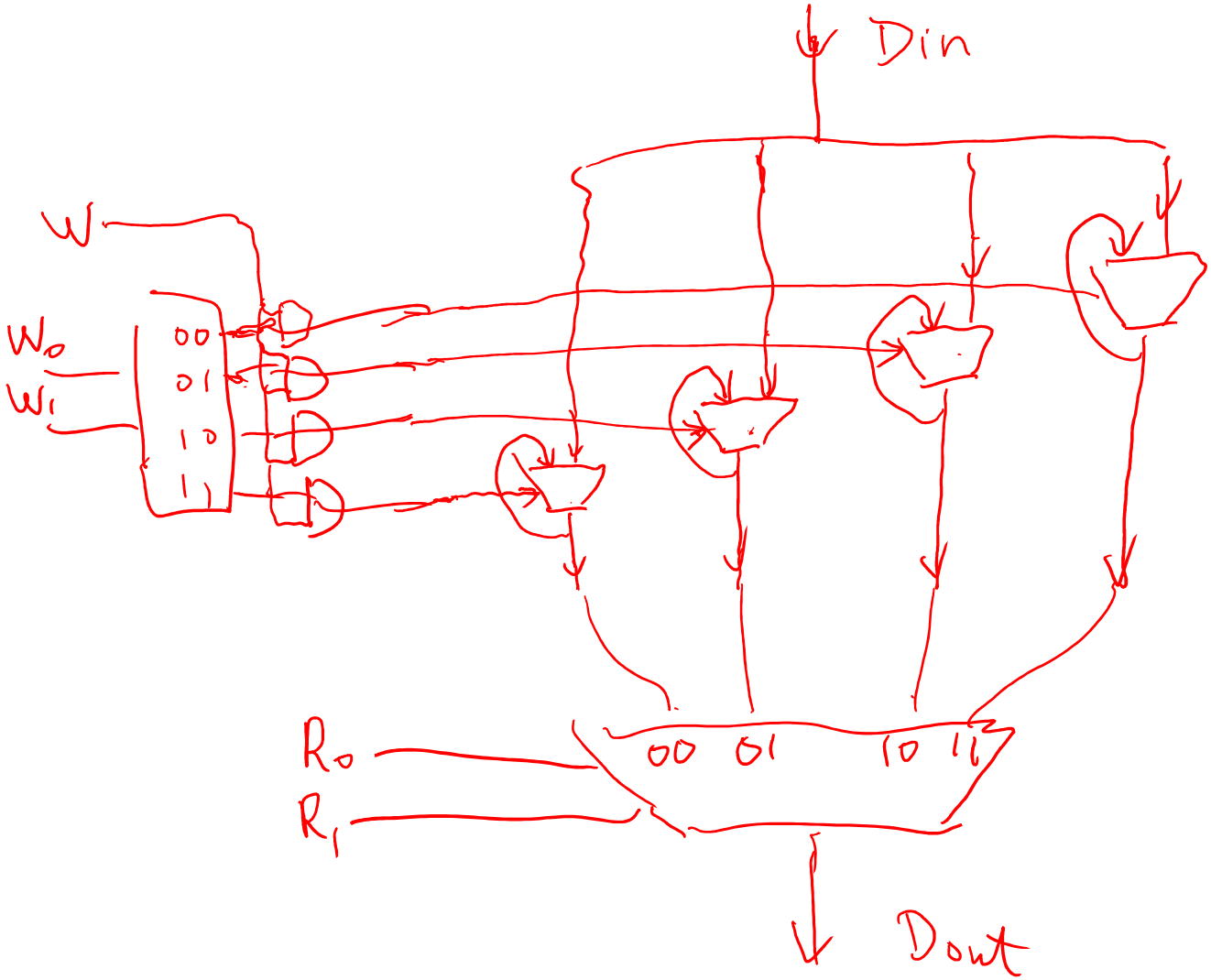
- read
- write
- Addressable

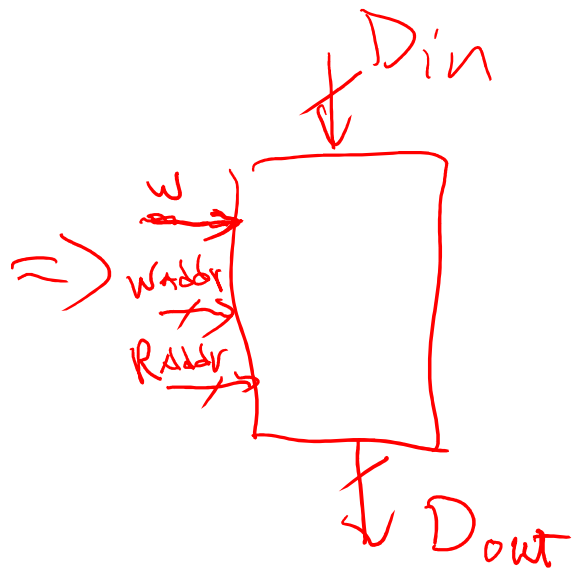
Sequential logic
v.s
combinational logic

A	B	S_0	Y
A	X	0	A
X	B	1	B

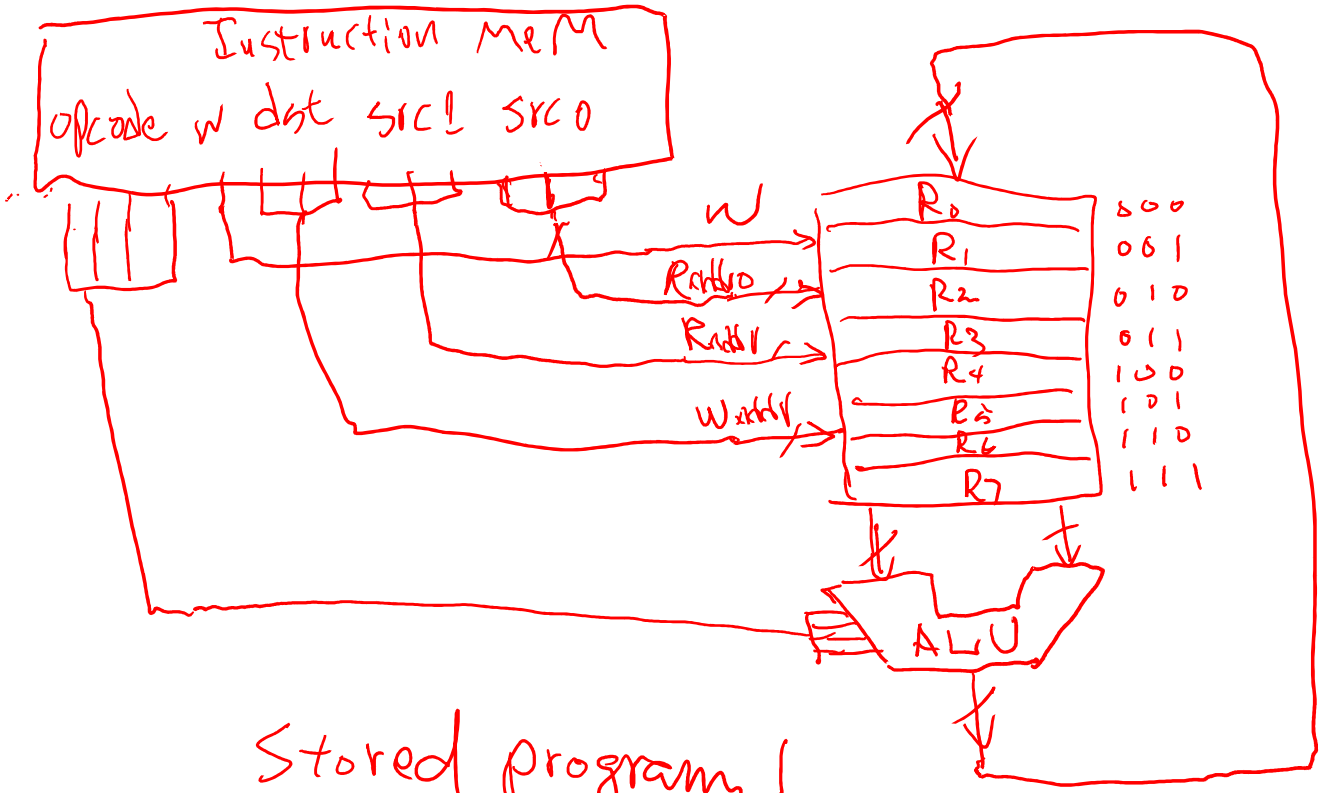
⇒

B	S_0	Y
0	0	?
1	0	?
1	1	1
1	0	1
0	0	1
0	1	0
0	0	0





Byte addressed



Register file

operands

Assembly code

Stored program!

Von Neumann

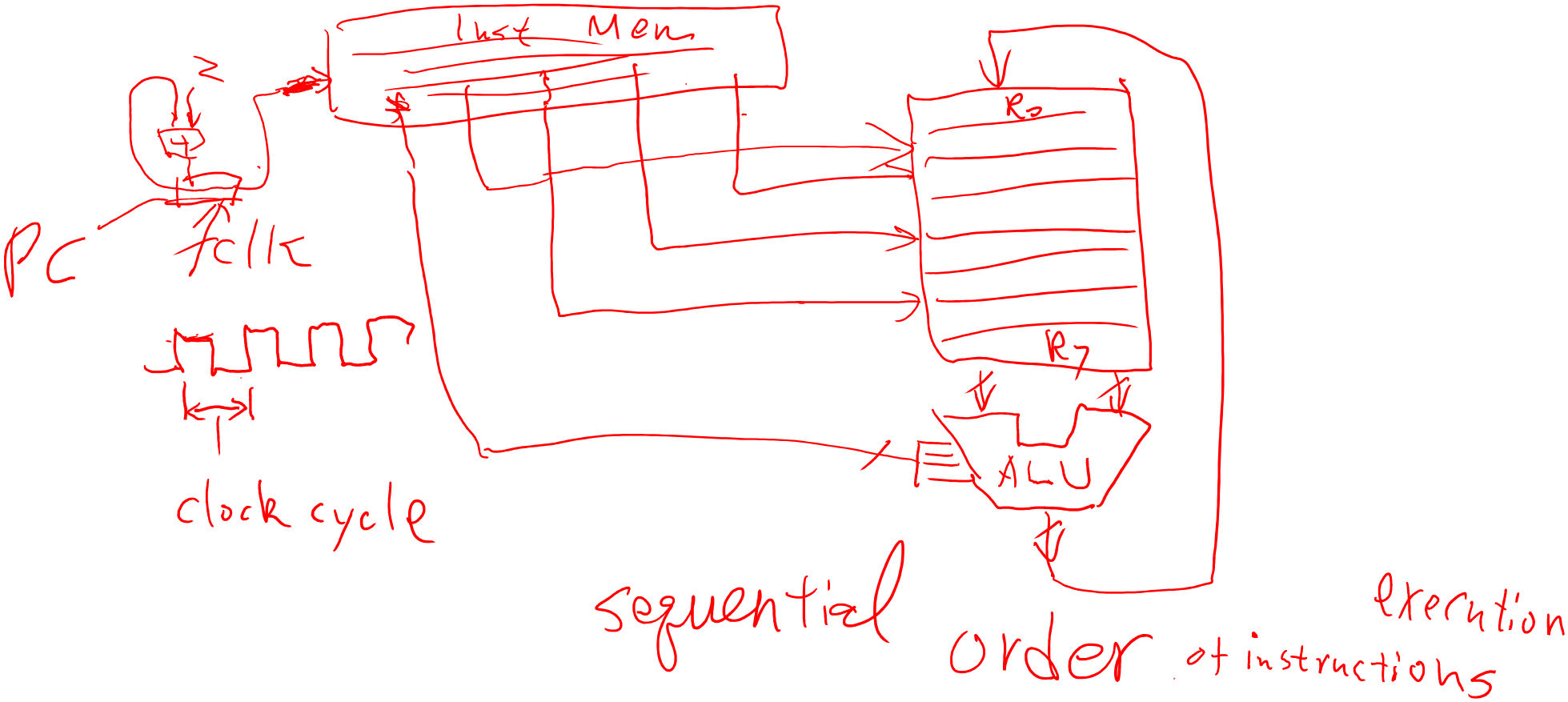
0000 1 111 001 000

opcode

Machine Instruction

ADD R_7, R_0, R_1

$R_7 \leftarrow R_0 + R_1$


Inst Mem
PC
fclk
clock cycle
R0
R7
ALU
sequential
order
execution
of instructions

Programming: Unbounded computation with finite size of program

Example: to compute c = a x b in a machine that does not have build-in instruction for multiplication

<u>Solution1:</u>

R0 ← a, R1 ← b // Load a, b into registers . Will discuss such instructions in ARM

```
SUB     R7, R7, R7
ADD     R7, R7, R0
ADD     R7, R7, R0
...
ADD     R7, R7, R0
```

# of instructions = b

Space complexity: O(b)

Time complexity: O(b)

Solution 2:

Register assignment: R0 ← a, R1 ← b, R7 ← c, R2 ← 1, R3 ← 0

Address:	Instruction	
0x00	**SUB**	**R7, R7, R7**
LOOP:	**ADD**	**R7, R7, R0**
0x04	**SUB**	**R1, R1, R2**
0x06	**BRZ**	**R1, EXIT**
0x08	**BRZ**	**R3, LOOP**
EXIT:		

```
c  = 0
while (b > 0) {
    c = c + a;
    b = b - 1;
}
```

Default order of execution is sequential. We need a new type of instruction for flow control

Label: Address for label
LOOP: 0x02
EXIT: 0x0A

Space complexity: O(1) fixed size
Time complexity: O(b)

Branch Instruction (flow control)

Branch condition BAddr

If(condition true)
PC ← BAddr
Else
PC ← PC + 2 // sequence order, add 2 bytes to get
// the addr of the next instruction

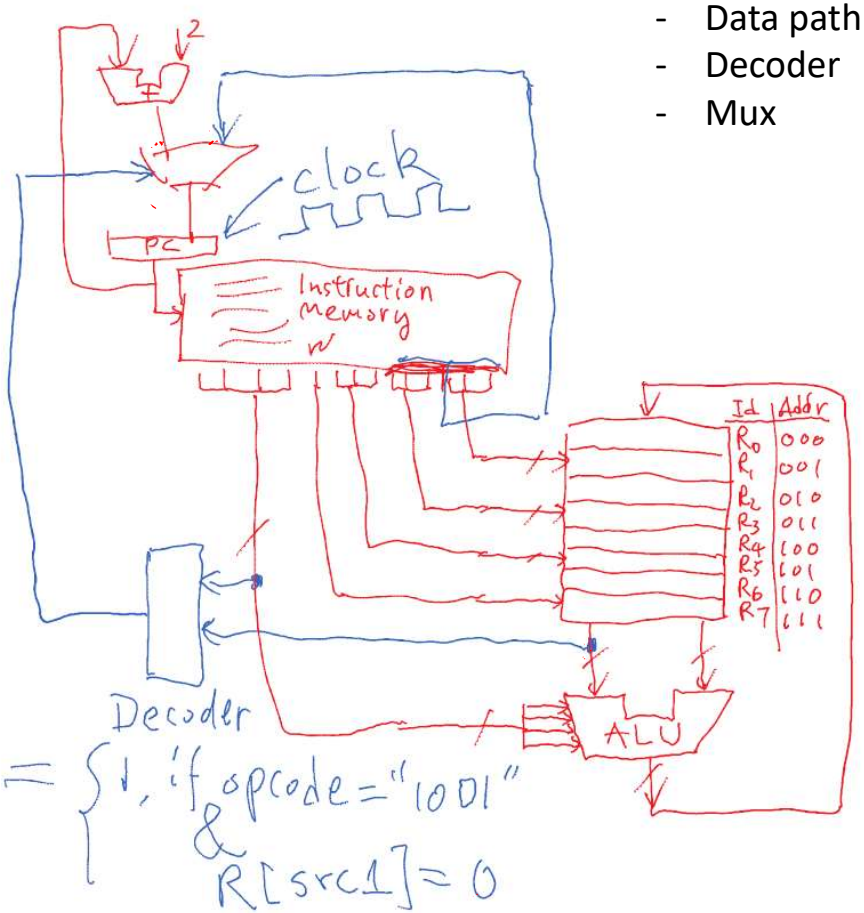
BRZ R3, LOOP

00 1001 0 011 000010 0X24C2

unused opcode w src BAddr

A Simple Computer

- Data path
- Decoder
- Mux



bits	15:14	13:10	9	8:6	5:3	2:0
field	unused	opcode	w	src1	src2	dst

where the fields are defined as follows.

opcode :	operation to be performed by the processor
w:	write back ALU output to register file (1 = yes, 0 = no)
src1:	address of the first ALU operand in the register file
src2:	address of the second ALU operand in the register file
dst:	address in the register file where the output is written

The opcode and meaning of these instructions are listed in the following table.

opcode	Binary encoding	Operation
ADD	0x0	R[src1] + R[src2] → R[dst]
SUB	0x1	R[src1] – R[src2] → R[dst]
SLL	0x2	R[src1] << 1 → R[dst]
SRL	0x3	R[src1] >> 1 → R[dst]
INV	0x4	~R[src1] → R[dst]
XOR	0x5	R[src1] ^ R[src2] → R[dst]
OR	0x6	R[src1] | R[src2] → R[dst]
AND	0x7	R[src1] & R[src2] → R[dst]
INCR	0x8	R[src1] + 1 → R[dst]
BRZ	0x9	If R[src1] = 0, branch to BAddr
BLEZ	0xA	If R[src1] ≤ 0, branch to BAddr
JUMP	0xE	Jump to JAddr
HALT	0xF	Stop execution

Solution 3:

Register assignment: R0 ← a, R1 ← b, R7 ← c, R2 ← 1, R3 ← 0

Source code

```
c  = 0
while (b > 0) {
   if(b & 0x01 == 1)
      c = c + a;
   a = a << 1;
   b = b >> 1;
}
return c;
```

Assembly code

Address:	Instruction	
0x00	**SUB**	**R7, R7, R7**
LOOP:	**BRZ**	**R1, EXIT**
0x04	**AND**	**R4, R1, R2**
0x06	**BRZ**	**R4, ELSE**
0x08	**ADD**	**R7, R7, R0**
ELSE:	**SLL**	**R0, R0**
0x0C	**SRL**	**R1, R1**
0x0E	**BRZ**	**R3, LOOP**
EXIT:	**HALT**	

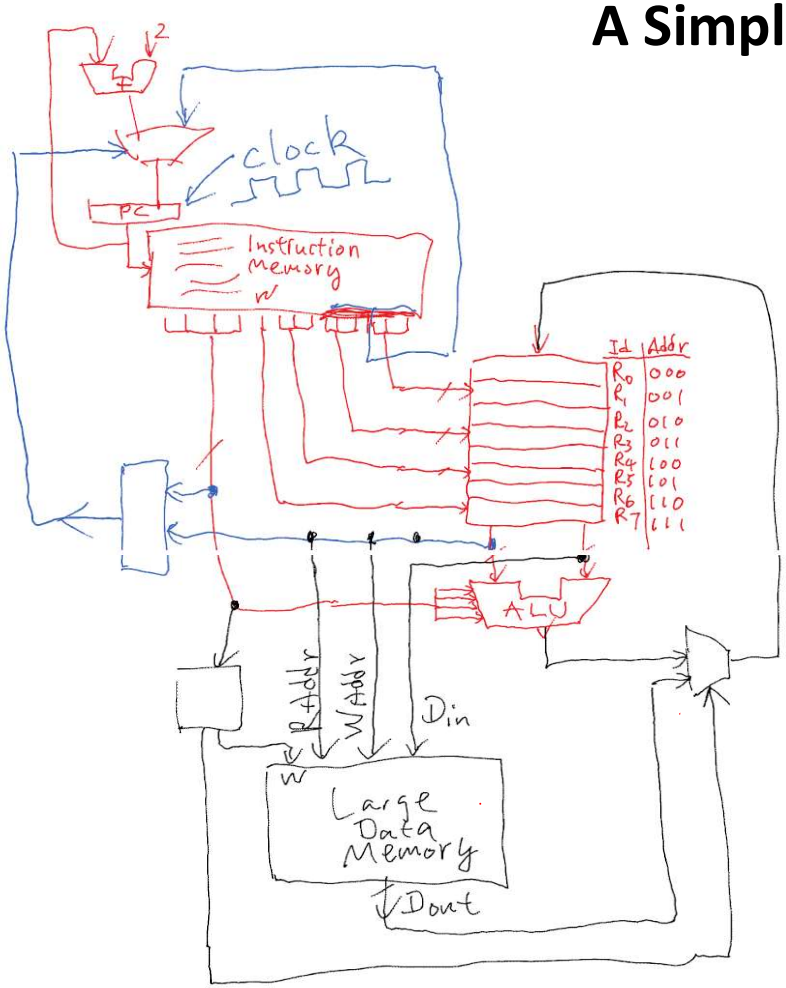
Machine code

unused	opcode	w	src1	src2	dst
00	0001	1	111	111	111
00	1001	0	001	010000	
00	0111	1	001	010	100
00	1001	0	100	001010	
00	0000	1	111	000	111
00	0010	1	000	000	000
00	0011	1	001	000	001
00	1001	0	011	000010	
00	1111	0	000	000	000

Space complexity: O(1)

Time complexity: $O(\log_2 b)$

A Simple Computer with added large memory



Two new instructions (data transportation):

LD src1, dst

R[dst] ← M[R[src1]]

ST src1, src2

M[R[src1]] ← R[src2]

- Data path
- Decoder
- Mux

Instruction independent from data, indirect: operand is not the data itself, but the location of the data.