Computer Systems and Networks

ECPE 170 – Jeff Shafer – University of the Pacific

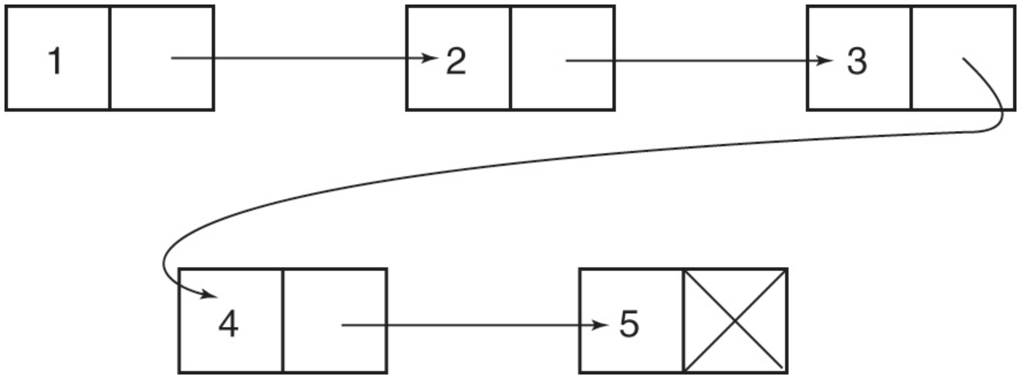
MARIE Instruction Decoding

Schedule

- **Today**
  - MARIE instruction decoding hardware
  - Plus **Quiz 3!**
- **Thursday** – Begin Chapter 5
  - Closer look at instruction sets

Homework 4.33 Review

```
Addr,  Hex ____        / Top of list pointer
Node2, Hex 0032        / Node's data is the character "2."
       Hex ____        / Address of Node3.
Node4, Hex 0034        / Character "4."
       Hex ____
Node1, Hex 0031        / Character "1"
       Hex ____
Node3, Hex 0033        / Character "3"
       Hex ____
Node5, Hex 0035        / Character "5"
       Hex 0000        / Indicates terminal node.
```



Instruction Decoding

Processor Control Unit

- Role of processor **control unit**
  - Keeps operations synchronized
  - Make sure that bits flow to the correct components at the correct time
- How can we build this control unit?
  - **Hardwired control**, or
  - **Microprogrammed control**
- The result is the same – control signals!

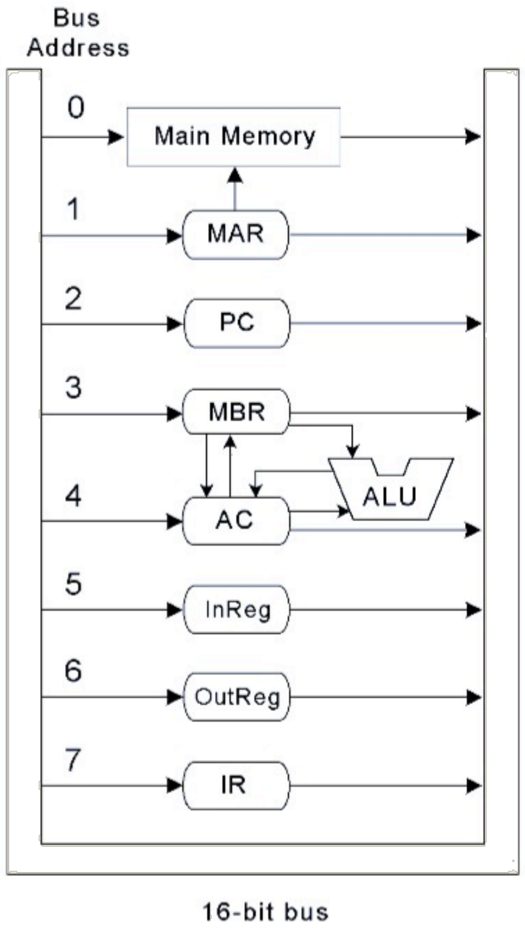
Processor Control Unit

- Remember the register transfer language description of each MARIE instruction?
  - *Table 4.7*
  - This is what the control unit manages
- Each microoperation consists of a distinctive signal pattern that is interpreted by the control unit and results in the execution of an instruction
  - RTL for the Add instruction:

```
MAR ← X
MBR ← M[MAR]
AC ← AC + MBR
```

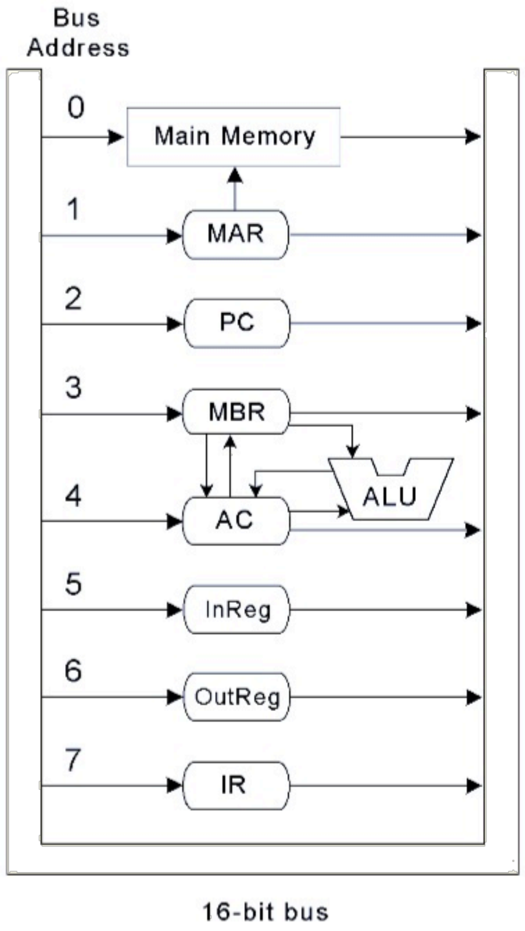
Processor Control Unit

- MARIE registers and main memory have a unique datapath address
- This address is issued as control signals by the control unit
- **How many signal lines does MARIE's control unit need to manage registers/main memory?**

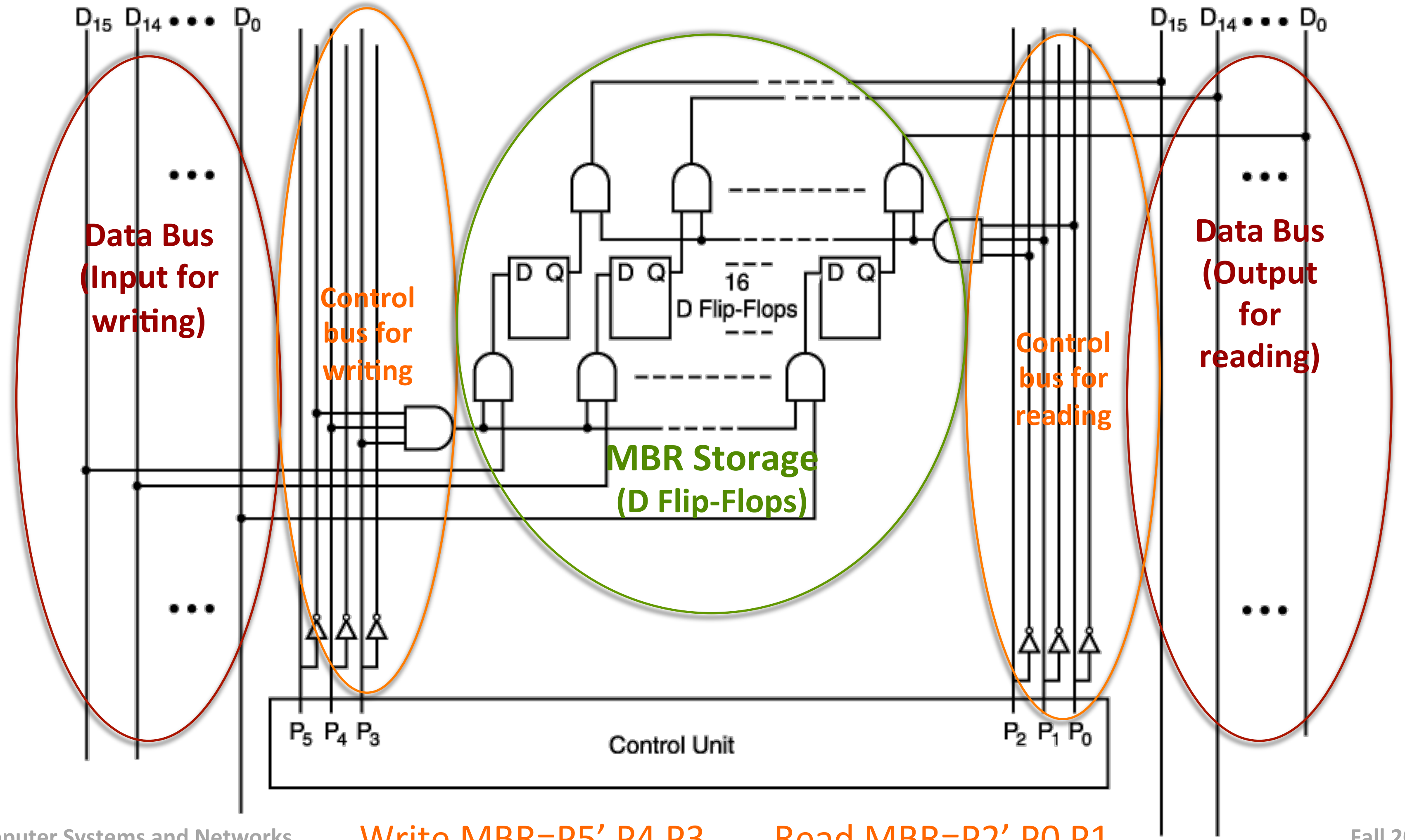


Processor Control Unit

- Two sets of three signals each
- **{P2, P1, P0}**
  - Controls **reading** from memory or a register
- **{P5, P4, P3}**
  - Controls **writing** to memory or a register
- What does this look like in detail?
  - **MBR shown next**



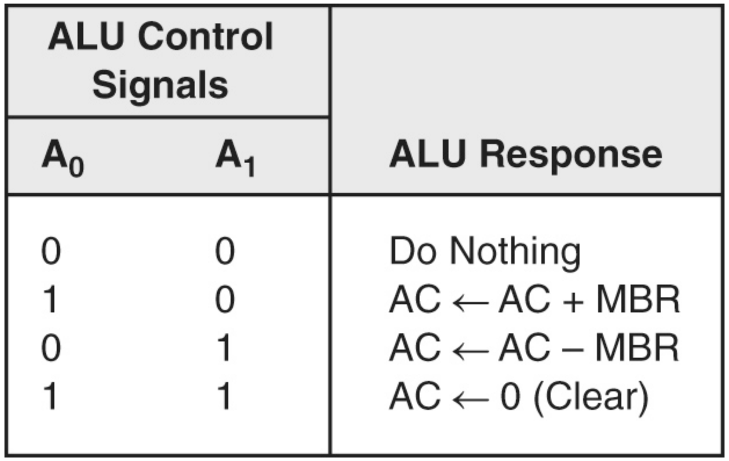
Memory Buffer Register (MBR) Closeup


D_{15} D_{14} ••• D_0
Data Bus (Input for writing)
Control bus for writing
D Q
16 D Flip-Flops
MBR Storage (D Flip-Flops)
Control bus for reading
D_{15} D_{14} ••• D_0
Data Bus (Output for reading)
P_5 P_4 P_3
Control Unit
P_2 P_1 P_0

Write MBR=P5’ P4 P3 Read MBR=P2’ P0 P1

Processor Control Unit

- Control unit must manage more than just registers/ main memory
  - What about the ALU modes?
- ALU has only four operations
  - Add, subtract, clear, and "do nothing"
- **ALU controls: A_0 – A_1**



ALU Control Signals		
A_0	A_1	ALU Response
0	0	Do Nothing
1	0	$AC \leftarrow AC + MBR$
0	1	$AC \leftarrow AC - MBR$
1	1	$AC \leftarrow 0$ (Clear)

Processor Control Unit

- How does the control unit perform operations in **sequence?**
- Longest instruction is JNS (look at RTL in Table 4.7)
  - 7 steps
  - Need a 3-bit counter wired to a 3-8 decoder
  - Counter reset for shorter instructions
- **Output of decoder is "timing" signals: $T_0 – T_7$**

- **The entire set of MARIE's control signals consists of:**
  - Register controls
    - P_0 through P_5
  - ALU controls
    - A_0 through A_1
  - Timing
    - T_0 through T_7
  - Counter reset C_r

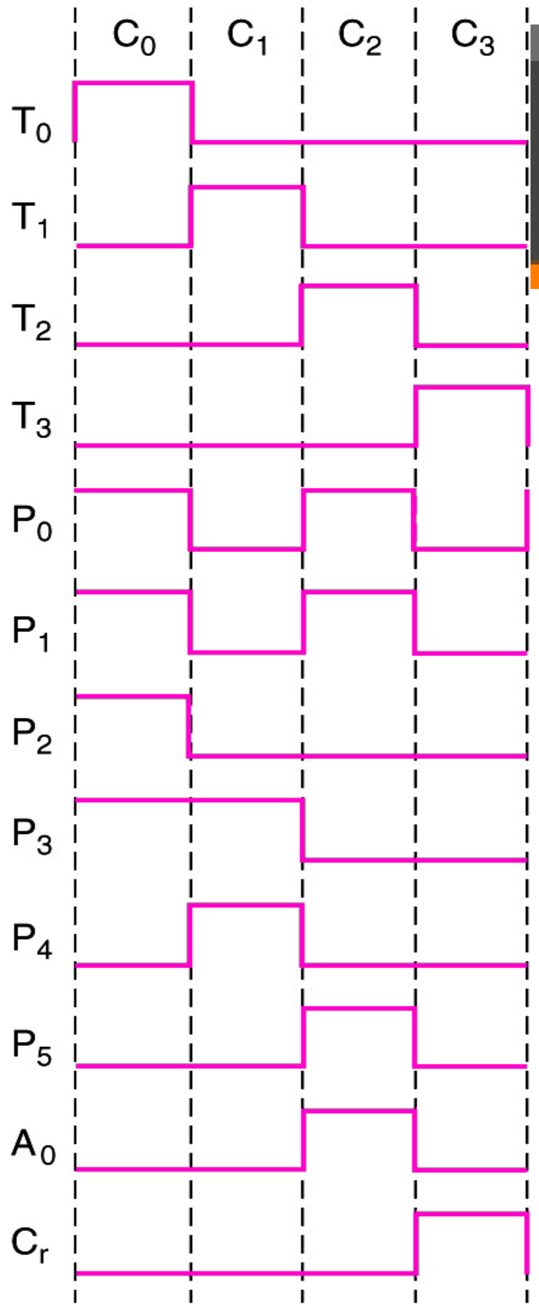
ADD Instruction Control

- ADD instruction RTL
  - `MAR ← X`
  - `MBR ← M[MAR]`
  - `AC ← AC + MBR`
- After the add instruction is fetched, the address (X) is in the rightmost 12 bits of the IR
  - IR datapath address is 7
  - Raise signals P2, P1, and P0 to **read from IR**
- X is copied to the MAR
  - MAR datapath address is 1
  - Raise signal P3 to **write to MAR**

ADD Instruction Control

- Complete signal sequence for ADD instruction
  - P3 P2 P1 P0 T0: MAR $\leftarrow$ X
  - P4 P3 T1: MBR $\leftarrow$ M[MAR]
  - A0 P5 P1 P0 T2: AC $\leftarrow$ AC + MBR
  - Cr T3: [Reset counter]
- These signals are ANDed with combinational logic to bring about the desired machine behavior

ADD Instruction Control



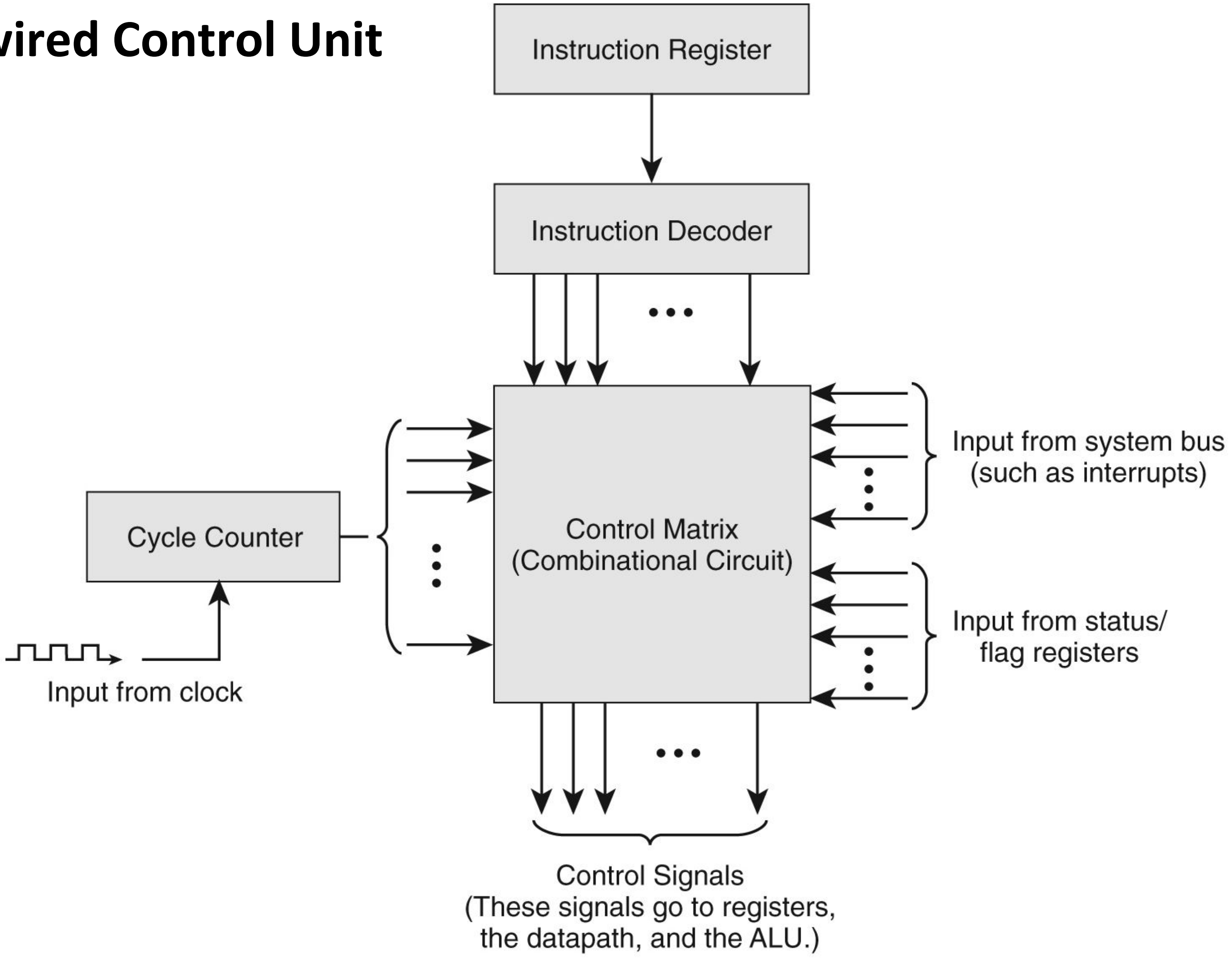
- Add instruction timing diagram
- Notice the concurrent signal states during each machine cycle: C0 through C3.

```
P3 P2 P1 P0 T0: MAR ← X
      P4 P3 T1: MBR ← M[MAR]
A0 P5 P1 P0 T2: AC ← AC + MBR
         Cr T3: [Reset counter]
```

Processor Control Unit

- This signal pattern needs to be produced regardless of whether the processor uses *hardwired* or *microprogrammed* control
- **Hardwired control unit**
  - Control unit is pure digital logic
- **Microprogrammed control unit**
  - A tiny program (called "microcode") saved in ROM
    - Even more rudimentary than assembly language!
  - Microinstructions are fetched, decoded, and executed in the same manner as regular instructions
  - Control unit works like a processor-in-miniature

Hardwired Control Unit



Hardwired vs Microprogrammed

Hardwired

- "Faster" (historically)
- Simple
- **Fixed** – requires redesigning circuit to change operation

All modern processors use some form of microprogramming for control

Microprogrammed

- "Slower" (historically - due to extra level of instruction interpretation)
- **Flexible** – Changing firmware alters the way processor executes instruction
  - If firmware is in flash, you can reprogram processor to fix bugs!
- **Scalable** – Supports complicated instructions with minimal hardware overhead

Quiz 3 – MARIE Stack

Simple Stack

- Operations:
  - PUSH
  - PEEK
  - POP
- Two key stack variables
  - StackBasePtr – Pointer to base of stack
  - StackCtr – Count of current number of elements in stack

Simple Stack

- Start of program
  - Nothing on the stack!
  - Base pointer points to base of stack
  - Counter is 0

	Address	Contents
	000	[[Program]]
	001	[[Program]]
	…	[[Program]]
StackBasePtr	100	102
StackCtr	101	0
	102	
	103	
	104	

Simple Stack

- **Push [55]** operation
- Results after push
  - Counter is now 1
  - Stack element 0 created
- Where does the element go?
  - Mem[102+0]

Value of counter before incrementing!

	Address	Contents
	000	[[Program]]
	001	[[Program]]
	…	[[Program]]
StackBasePtr	100	102
StackCtr	101	1
Stack[0]	102	55
	103	
	104	

Simple Stack

- **Push [66]** operation
- Results after push
  - Counter is now 2
  - Stack element 1 created

	Address	Contents
	000	[[Program]]
	001	[[Program]]
	…	[[Program]]
StackBasePtr	100	102
StackCtr	101	2
Stack[0]	102	55
Stack[1]	103	66
	104	

Simple Stack

- **Peek** operation
- Results after peek
  - Counter is unchanged
  - Stack is unchanged
- Element 66 is available
- Where did we find 66?
  - Mem[102+(2-1)]

	Address	Contents
	000	[[Program]]
	001	[[Program]]
	…	[[Program]]
StackBasePtr	100	102
StackCtr	101	2
Stack[0]	102	55
Stack[1]	103	66
	104	

Simple Stack

- **Pop** operation
- Results after pop
  - Counter is now 1
  - Stack is unchanged
- **Don't need to modify the stack in memory**
  - 66 can persist as garbage value beyond current top of stack

	Address	Contents
	000	[[Program]]
	001	[[Program]]
	…	[[Program]]
StackBasePtr	100	102
StackCtr	101	1
Stack[0]	102	55
	103	66
	104	