Will he not stop teaching Microcode?

COMP375

Computer Architecture and Organization

“Perseverance is not a long race; it is many short races one after the other.”

Walter Elliot

Online Textbook Reading

- Read chapter 5 on caching
- Read section 9.3 about Virtual Memory

2nd Exam

- The next exam in COMP375 will be on Friday, October 25, 2019

Simple CPU

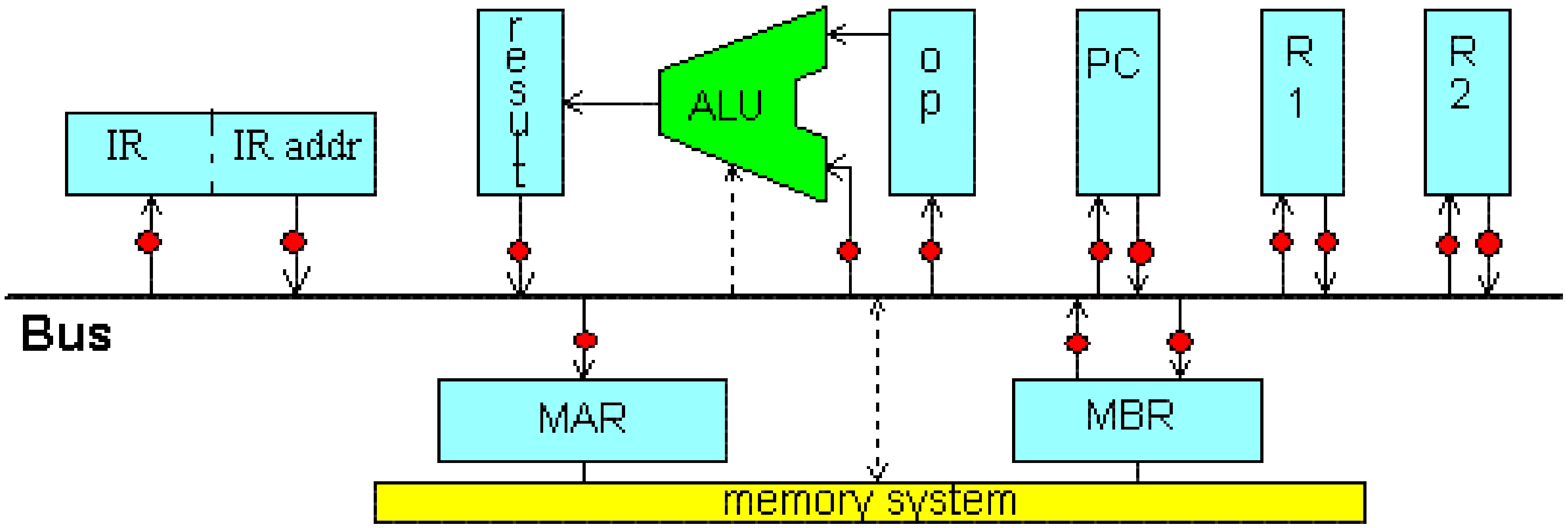
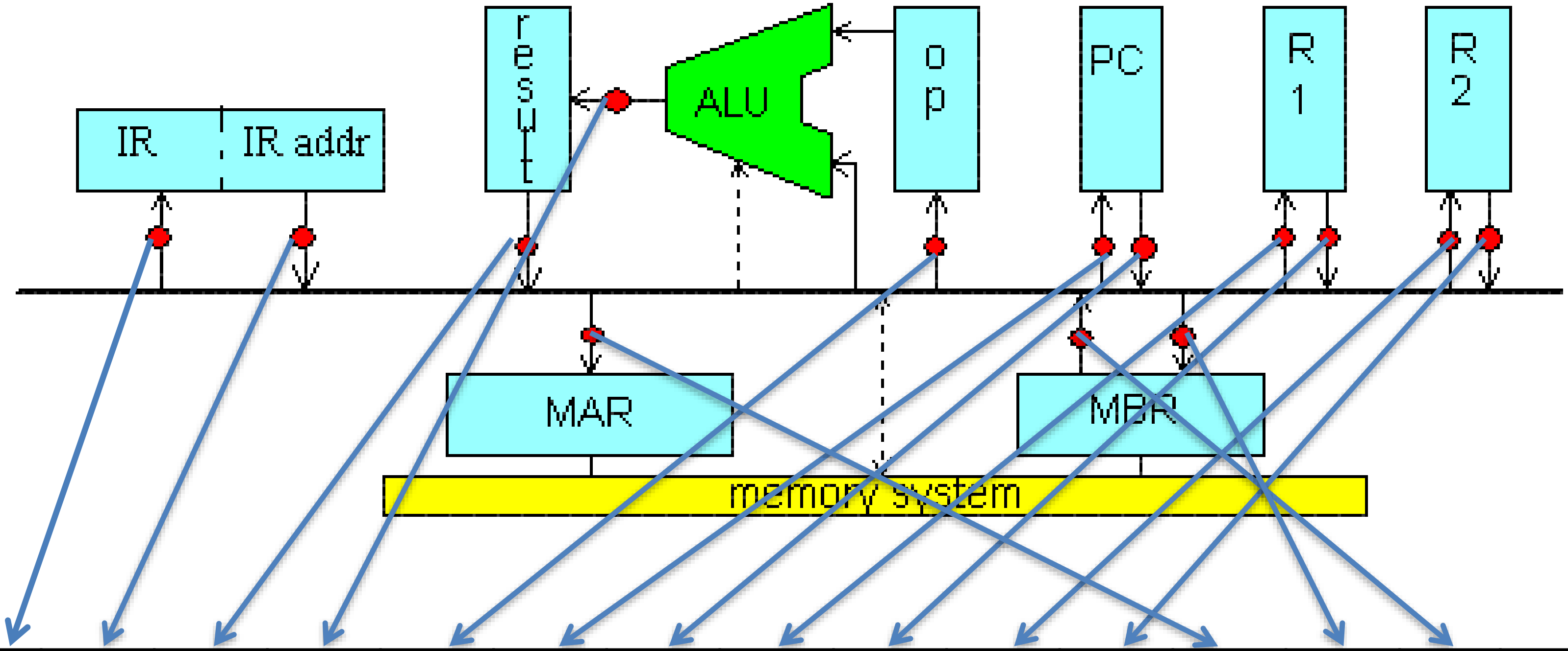


Table and Switches



bus → IR	IR adr → bus	result → bus	ALU → result	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU func	Mem func
	X				X										

Microcode Programming

- Think of microcode as programming
- “Coding” is writing software
- Microcoding is coding the hardware

New Microcode Format

- Use an assignment statement to represent a register transfer

```
R1 = result;
MAR = IRaddr;
```

- Allow multiple registers to the left

```
R2 = MAR = result;     // copies result to both R2 and MAR
MAR = R2 = result;     // order is unimportant
```

ALU function

- The ALU can appear like a register
- The ALU has an operand that specified the action
- ALU always updates the result register

```
ALU("inc") = R2;     // copy R2 to ALU and increment
ALU("add") = R1;     // add R1 value and operand register
```

- Binary ALU functions use the operand register

Memory Functions

- We create three memory functions

  read()

  write()

  wait()

- These are called to perform the obvious

Instruction Fetch and PC Increment

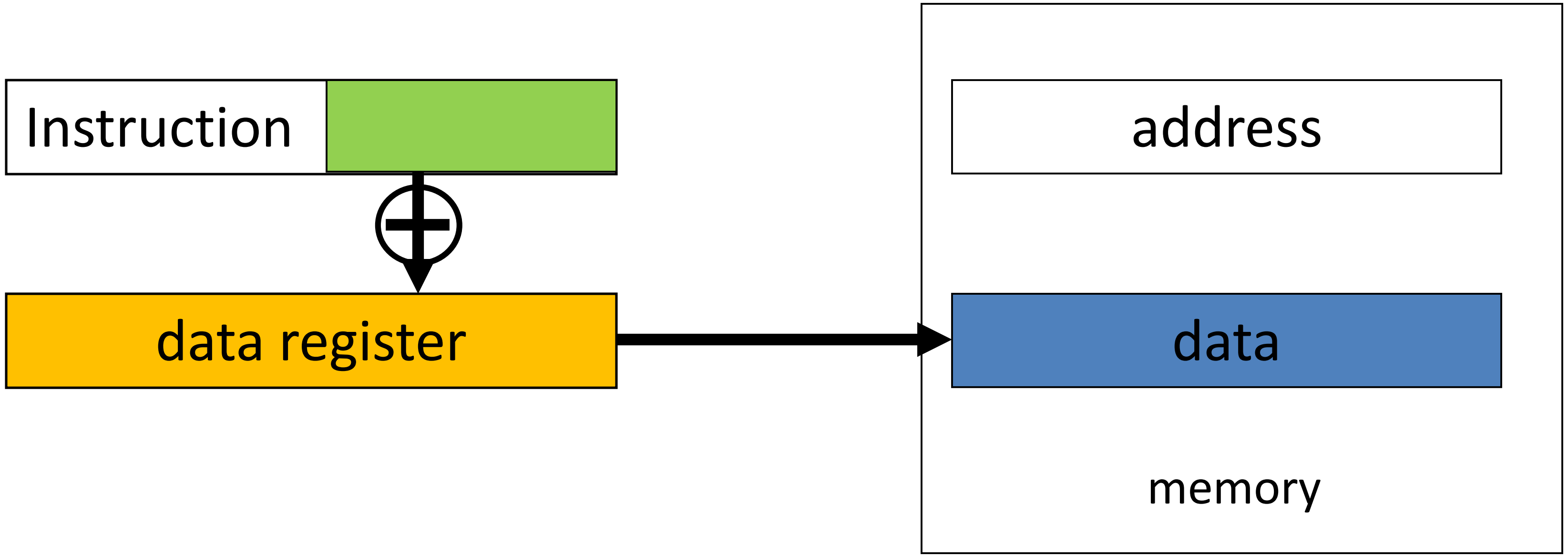
```
MAR = ALU("inc") = PC;
read();
PC = result;
wait();
IR = MBR;
```

The READ memory function is used

A. To get data from RAM
B. To get data from a register
C. All of the above
D. None of the above

Register Indirect with Offset

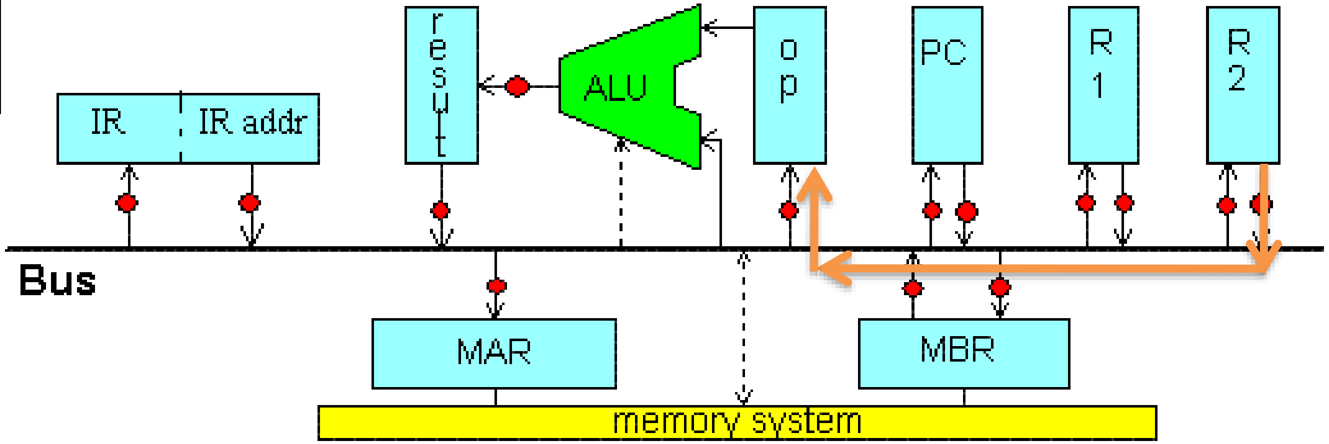
- The address of the data is the sum of the instruction offset field and a register value
- Useful when addressing an array



Register Indirect with Offset

Add to R1 using Register Indirect with Offset with R2 as the index

operand = R2;

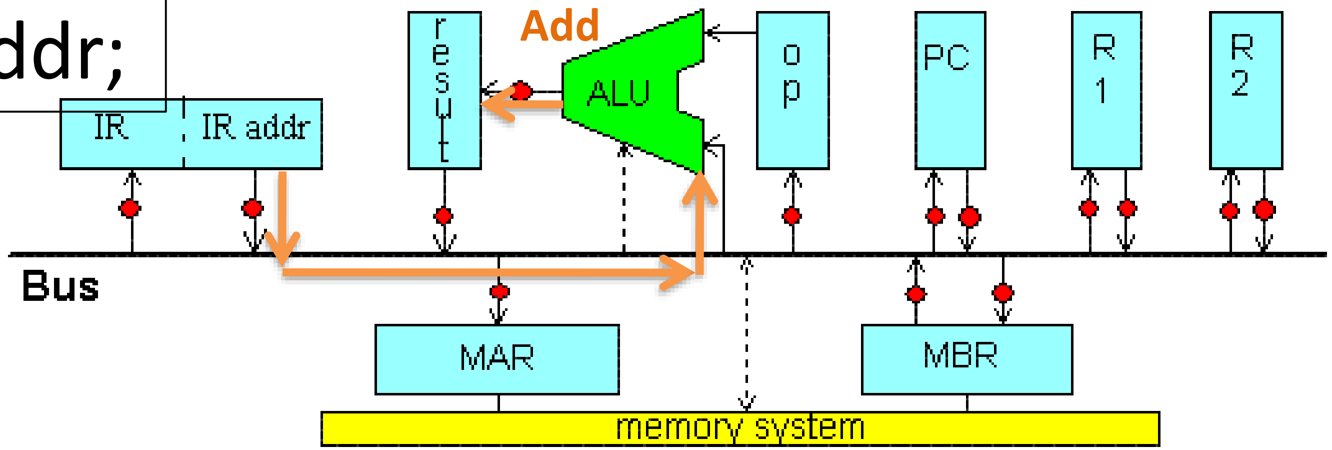


bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
				X						X					
	X		X											add	
		X									X				read
				X				X							wait
			X										X	add	
		X					X								

Register Indirect with Offset

Add to R1 using Register Indirect with Offset with R2 as the index

ALU("add") = IRaddr;

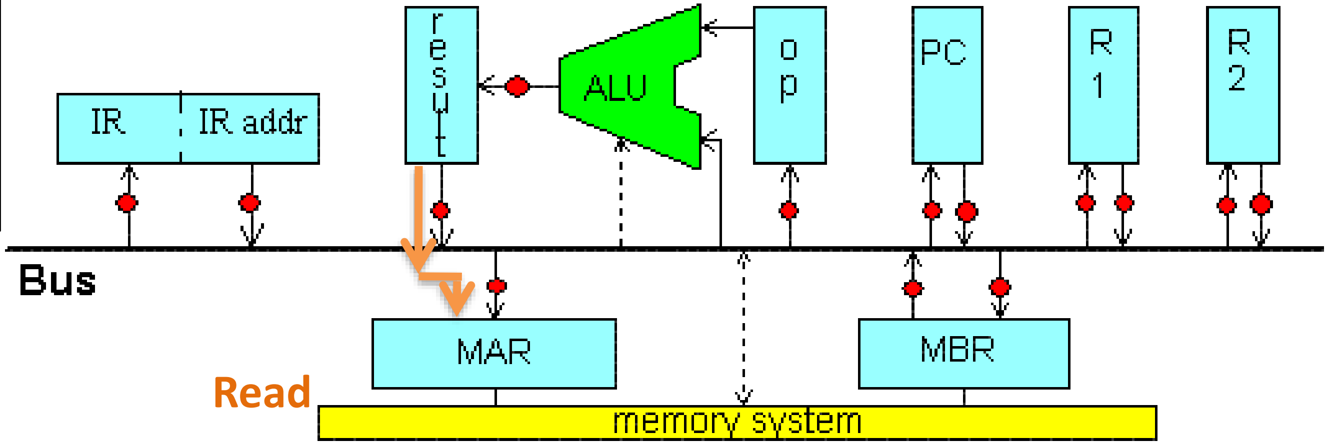


bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
				X						X					
	X		X											add	
		X									X				read
				X				X							wait
			X										X	add	
		X					X								

Register Indirect with Offset

Add to R1 using Register Indirect with Offset with R2 as the index

```
MAR = result;
read();
```

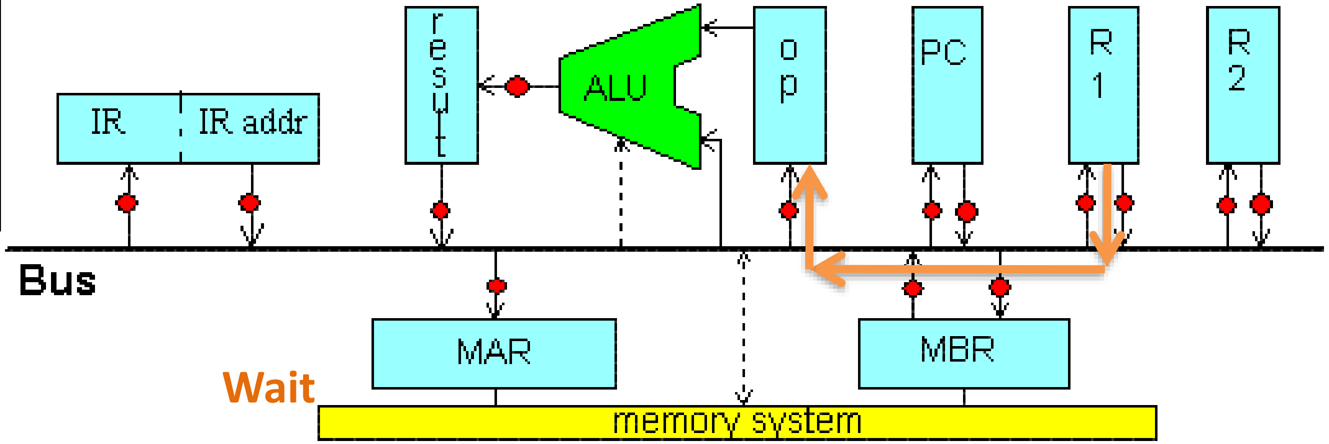


bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
				x						x					
	x		x											add	
		x									x				read
				x				x							wait
			x										x	add	
		x					x								

Register Indirect with Offset

Add to R1 using Register Indirect with Offset with R2 as the index

```
operand = R1;
wait();
```

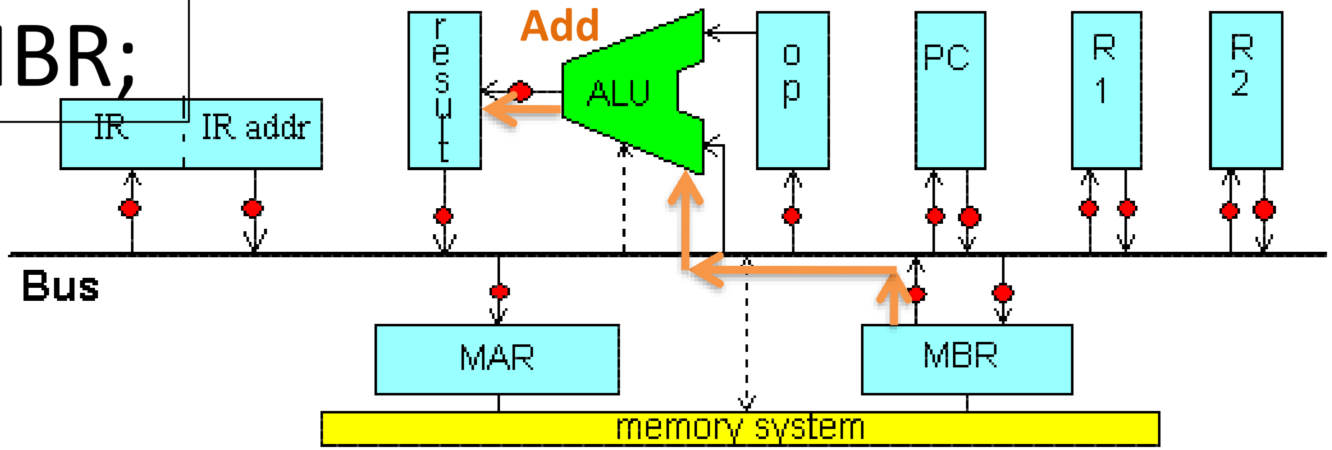


bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
				X						X					
	X		X											add	
		X									X				read
				X				X							wait
			X										X	add	
		X					X								

Register Indirect with Offset

Add to R1 using Register Indirect with Offset with R2 as the index

ALU("add") = MBR;

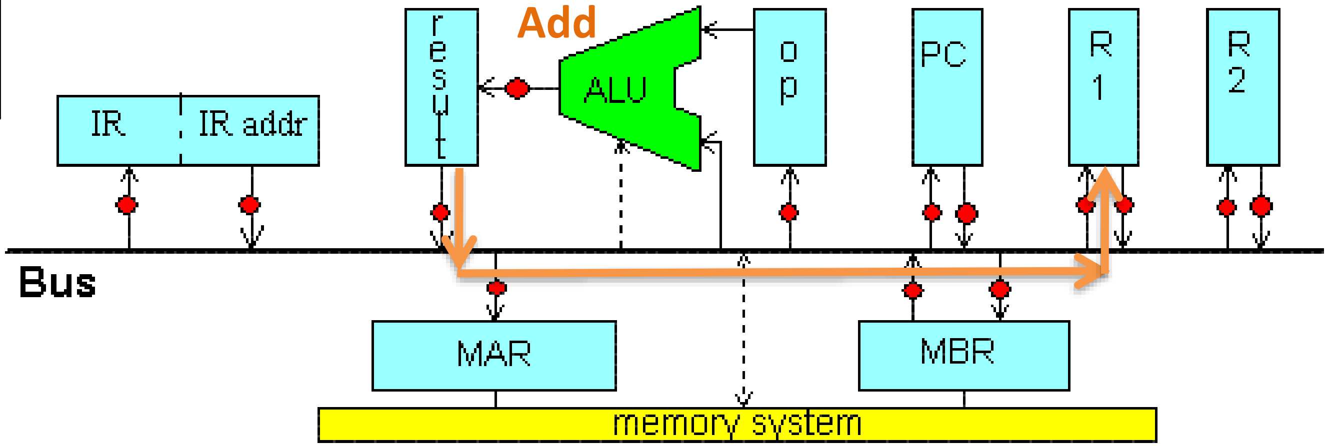


bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
				X						X					
	X		X											add	
		X									X				read
				X				X							wait
			X										X	add	
		X					X								

Register Indirect with Offset

Add to R1 using Register Indirect with Offset with R2 as the index

R1 = result;



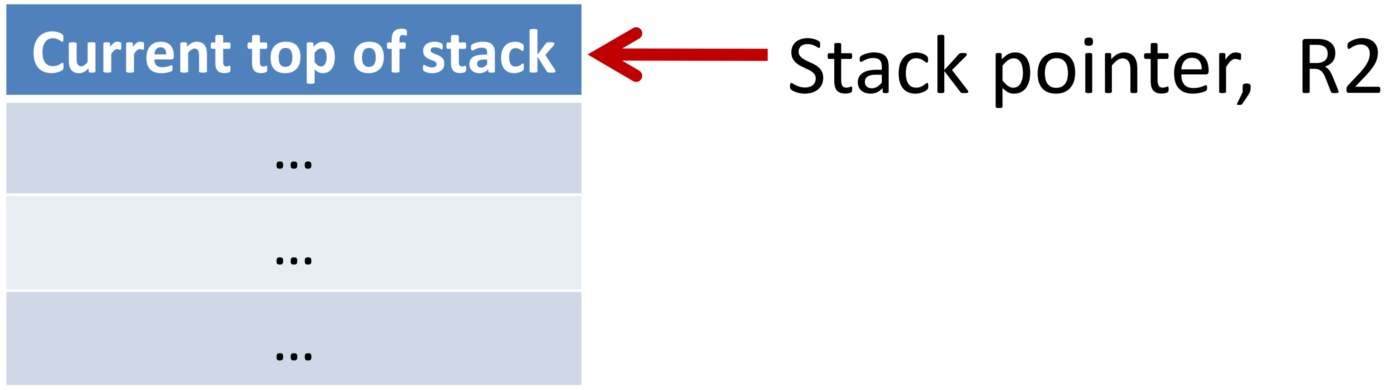
bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
				X						X					
	X		X											add	
		X									X				read
				X				X							wait
			X										X	add	
		X					X								

load R1, dog[R2]

```
operand = R2;
ALU("add") = IRaddr;
MAR = result;
read();  wait();
R1 = MBR;
```

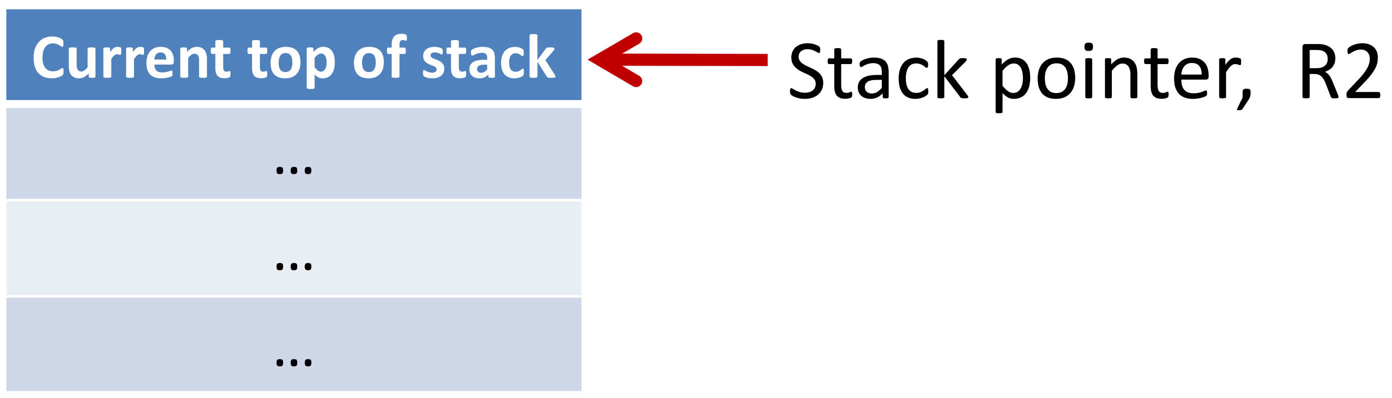
Pop Instruction

- The pop instruction reads the top of the stack into a register



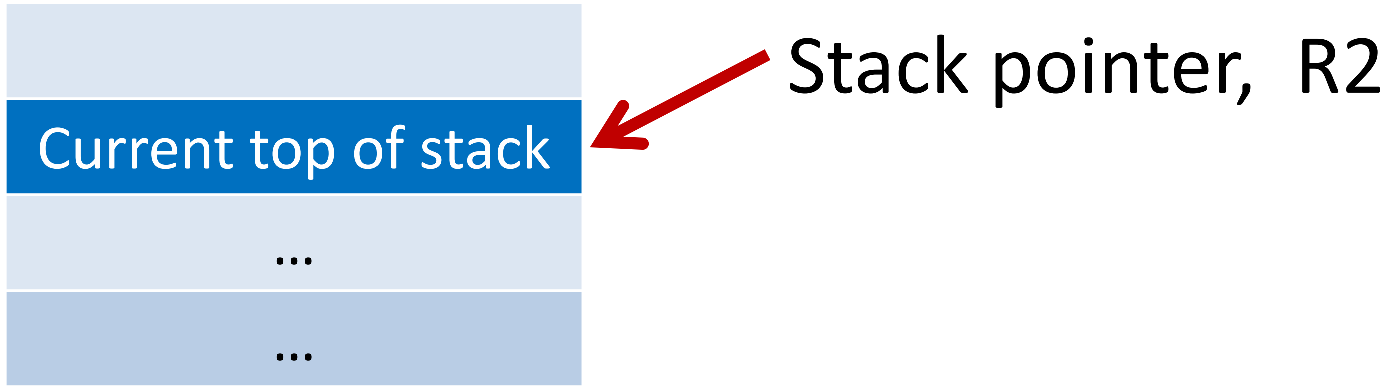
Pop Instruction

- Read the memory location whose address is in the stack pointer



Pop Instruction

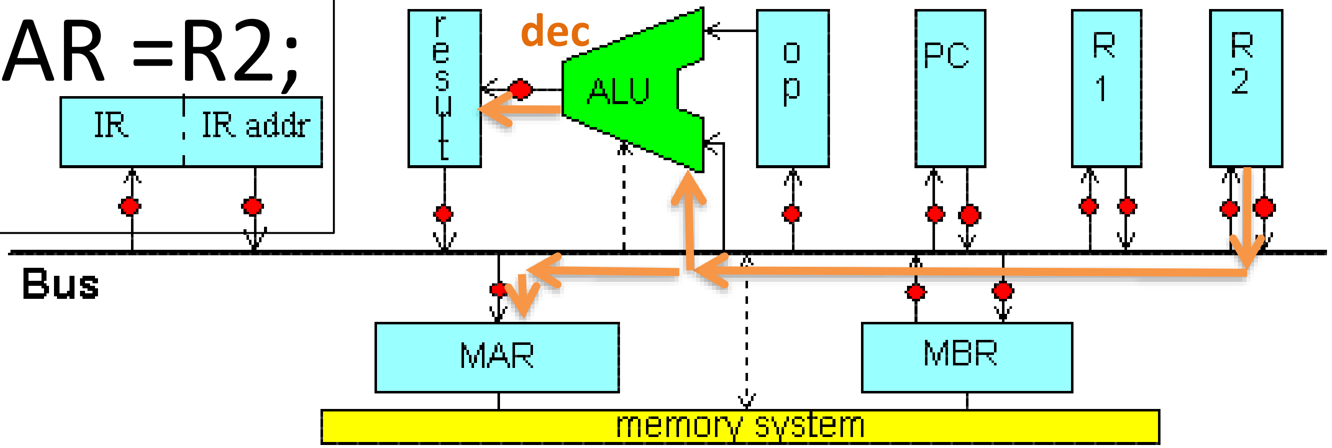
- Decrement the stack pointer



Pop R1 Assume R2 is the stack pointer

Read the address in R2 and store R1. Decremented R2.

ALU(“dec”) = MAR =R2;
read();

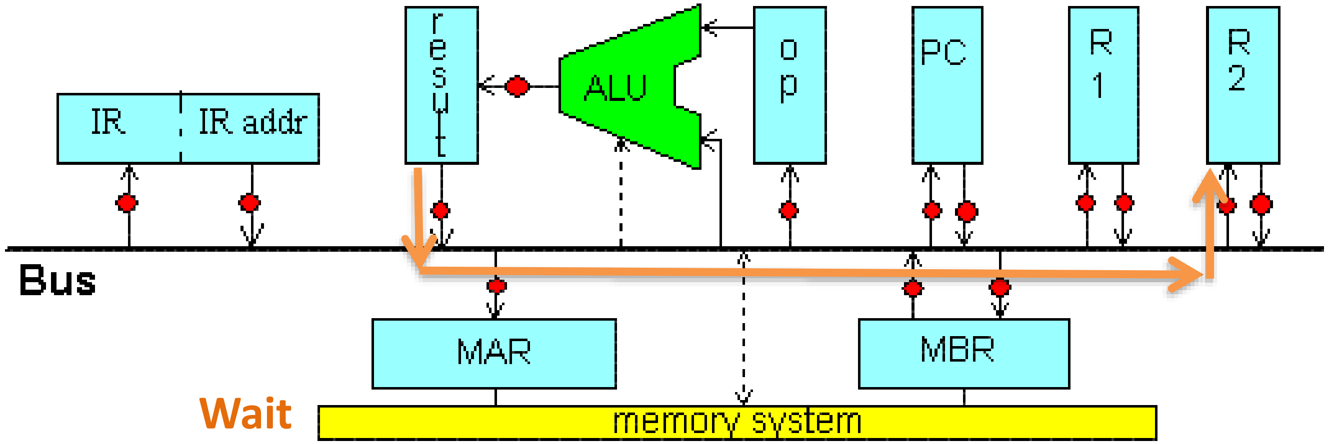


bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
			X							X	X			dec	read
		X							X						wait
							X						X		

Pop R1 Assume R2 is the stack pointer

Read the address in R2 and store R1. Decremented R2.

R2 = result;
wait();

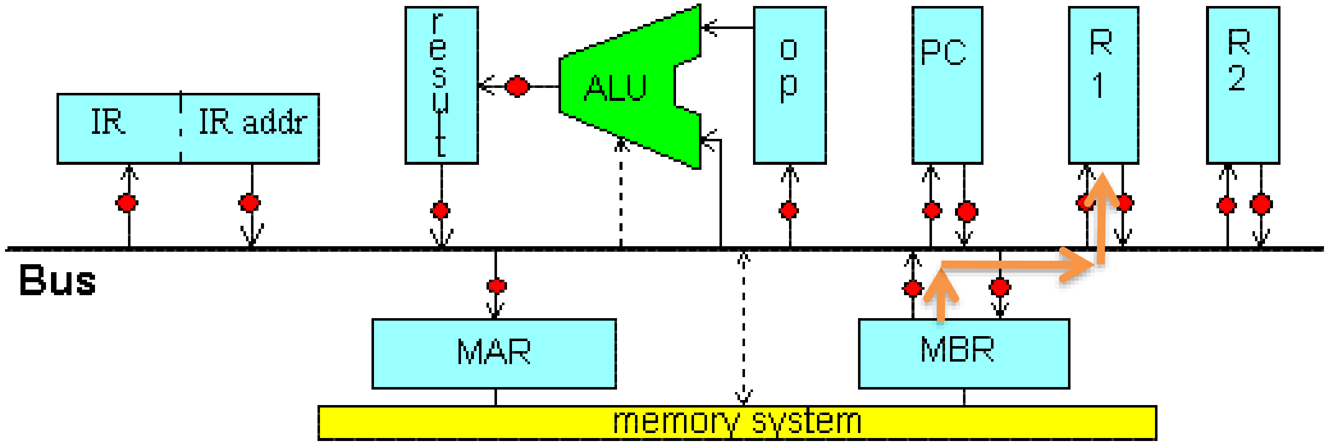


bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
			x							x	x			dec	read
		x							x						wait
							x						x		

Pop R1 Assume R2 is the stack pointer

Read the address in R2 and store R1. Decremented R2.

R1 = MBR;



bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
			x							x	x			dec	read
		x							x						wait
							x						x		

Pop R1 with R2 as Stack Pointer

```
ALU("dec") = MAR = R2;
read();
R2 = result;
wait();
R1 = MBR;
```

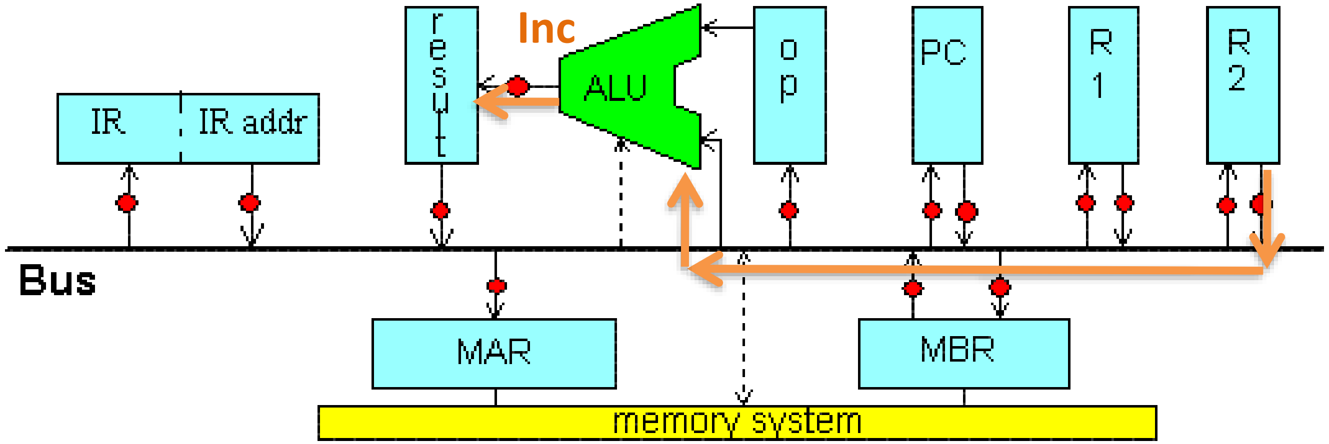
The ALU action is determined by the

A. Microcode
B. Opcode
C. All of the above
D. None of the above

Function Call

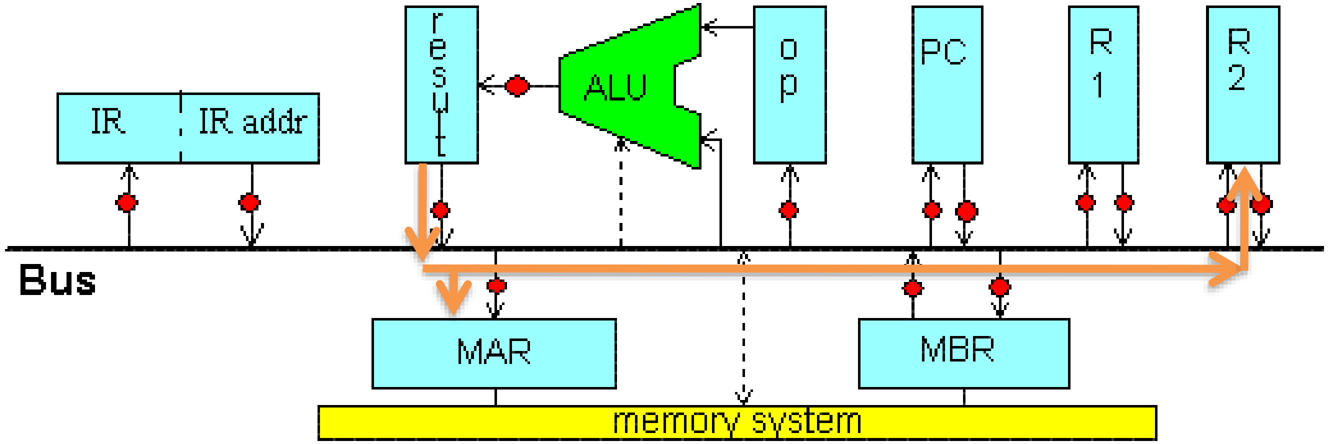
- A function call instruction pushes the return address on the stack and jumps to the start of the function
- The return address is in the Program Counter

CALL funcaddr - Push the return address on the stack and jump to the address of the function. R2 is stack ptr



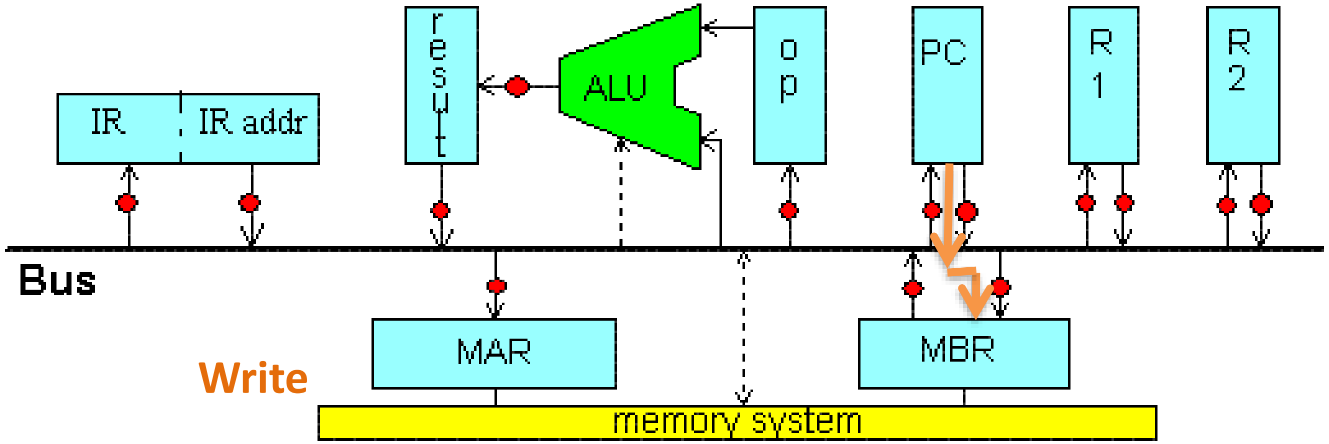
bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
			x							x				inc	
		x							x		x				
						x						x			write
	x				x										wait

CALL funcaddr - Push the return address on the stack and jump to the address of the function. R2 is stack ptr



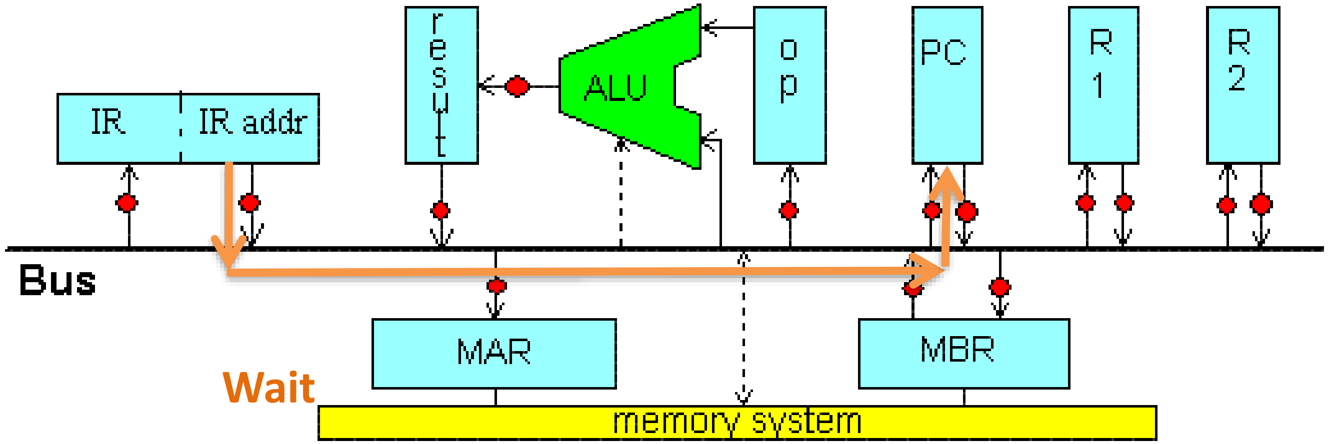
bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
			X							X				inc	
		X							X		X				
						X						X			write
	X				X										wait

CALL funcaddr - Push the return address on the stack and jump to the address of the function. R2 is stack ptr



bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
			x							x				inc	
		x							x		x				
						x						x			write
	x				x										wait

CALL funcaddr - Push the return address on the stack and jump to the address of the function. R2 is stack ptr



bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
			X							X				inc	
		X							X		X				
						X						X			write
	X				X										wait

Try It

- Write microcode to implement a method return
- Pop the address from the stack and put it in the program counter

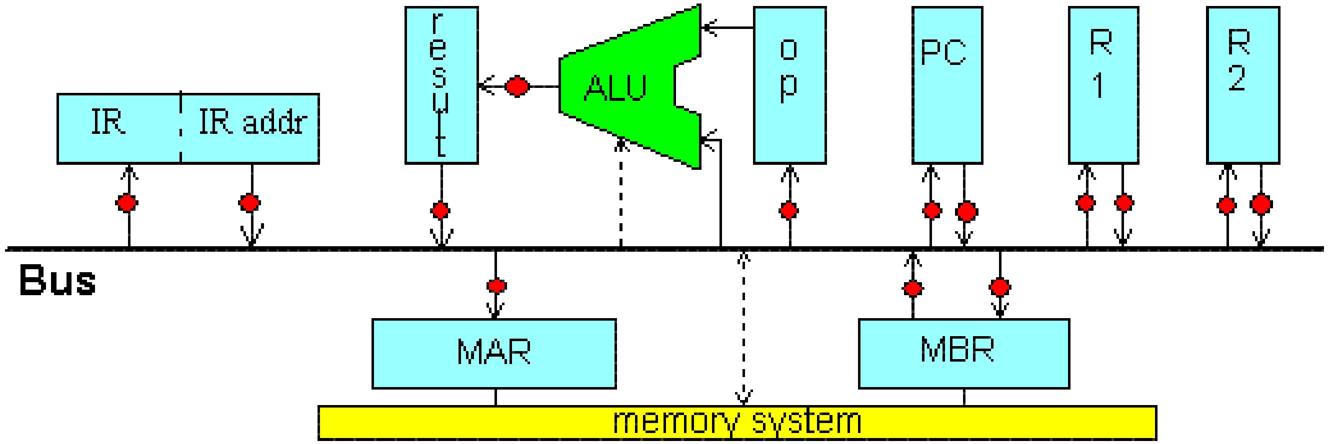
- Write the microcode in the programming style

Possible Solution

- Stack function return

```
ALU("dec") = MAR = R2;
read();
R2 = result;
wait();
PC = MBR;
```

`return` R2 is stack pointer



bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
			x							x	x			dec	read
		x							x						wait
					x								x		

Is the Programming Style better than the table?

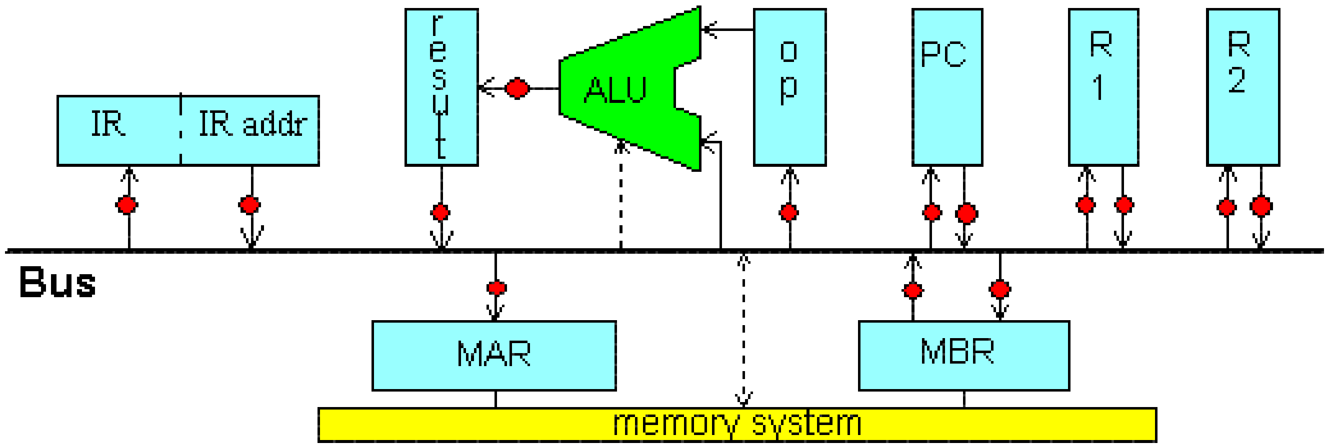
A. The programming style is better
B. The table of Xs is better
C. Both are about the same
D. I'm just going to flunk COMP375

Register Based Method Call

- Some systems do not use a stack for method calls
- The return address can be saved in a register
- Write the microcode for

```
call  R2, mymethod
```

Register Call



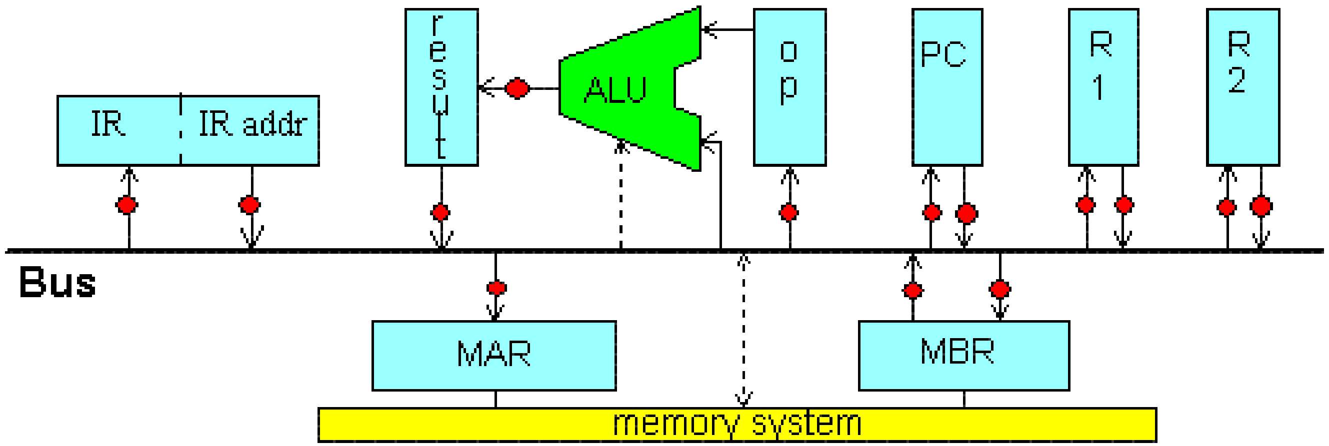
bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
						x			x						
	x				x										

Return for Register Based Call

- Write the microcode for a return instruction in a register based call system

```
return  R2
```

Register Return



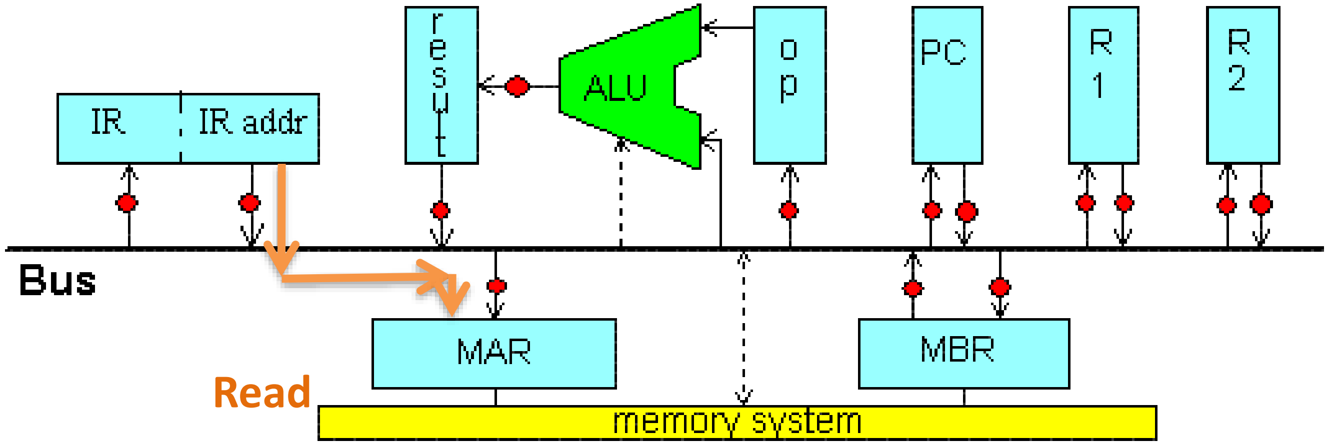
bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
						x				x					

Memory Operand Instruction

```
inc dog
```

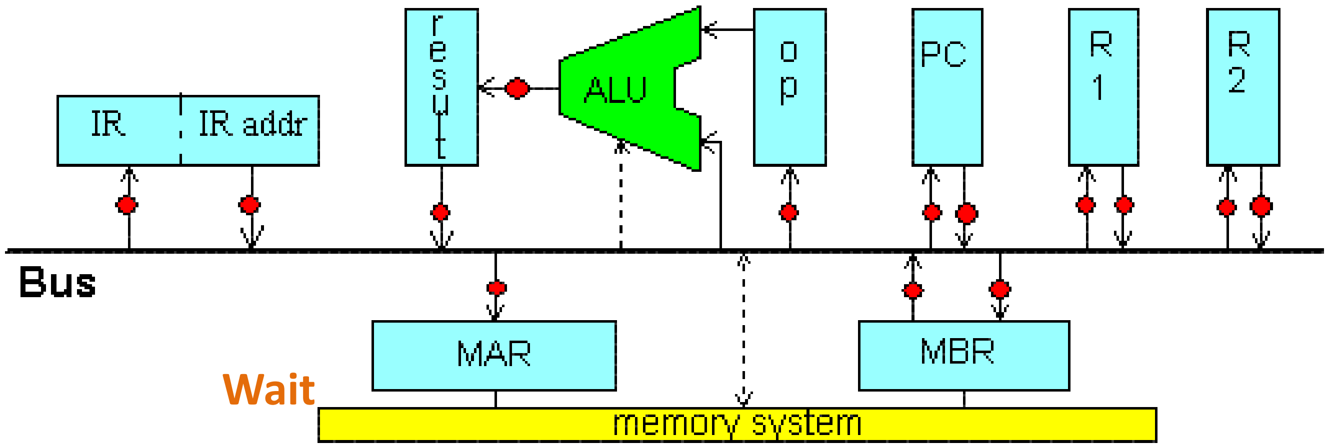
- The value at the memory location (dog) is incremented by one.
- Direct memory addressing
- Calculations occur in the CPU
  - Read memory value
  - Increment number
  - Store value back to memory

inc dog



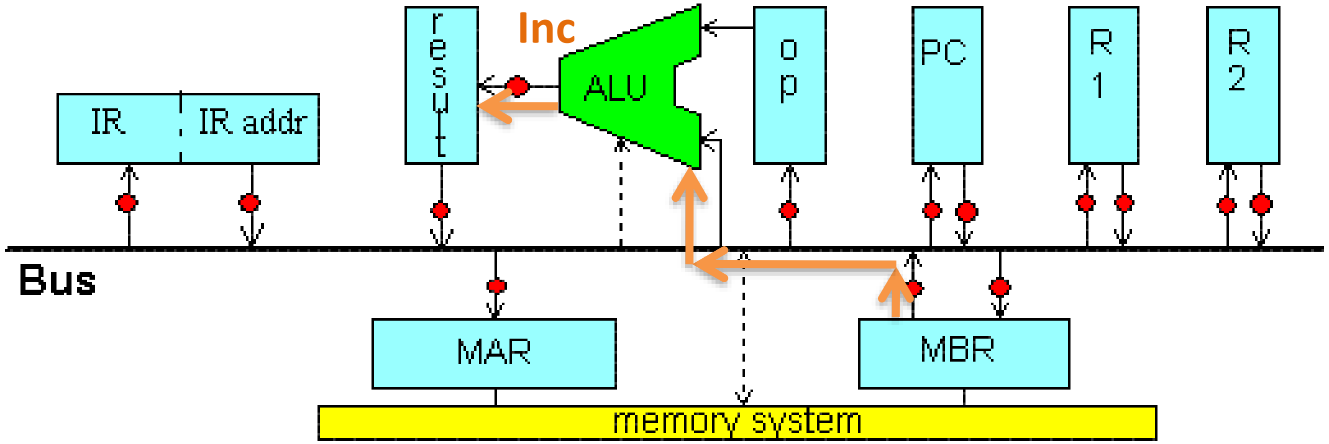
bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
	X										X				read
															wait
			X										X	inc	
		X										X			write
															wait

inc dog



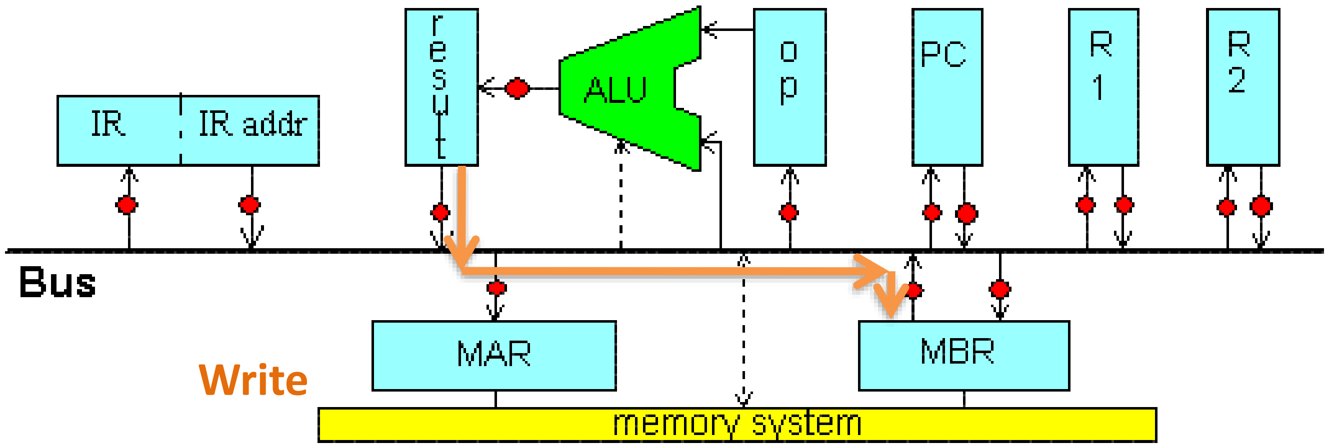
bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
	X										X				read
															wait
			X										X	inc	
		X										X			write
															wait

inc dog



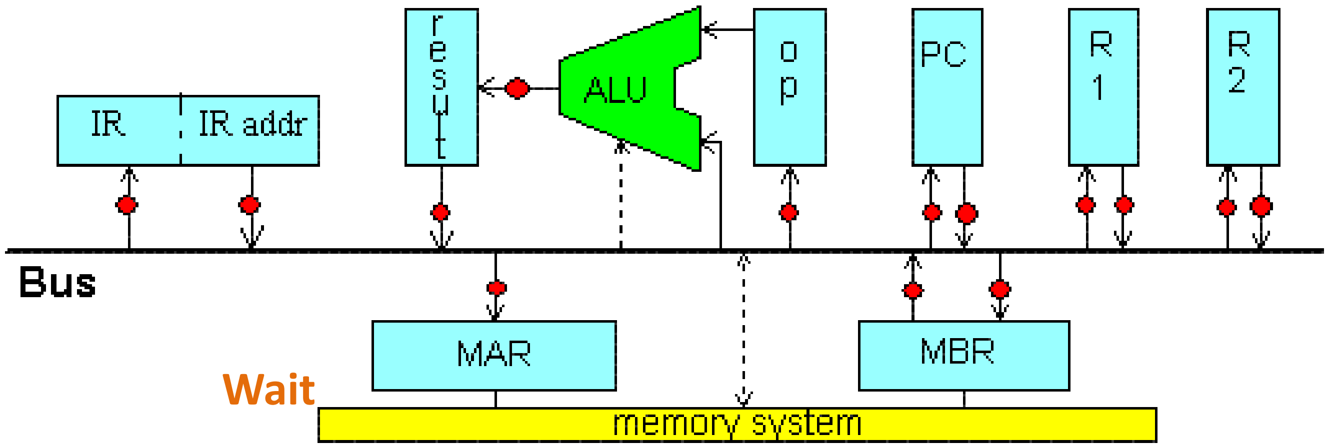
bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
	X										X				read
															wait
			X										X	inc	
		X										X			write
															wait

inc dog



bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
	X										X				read
															wait
			X										X	inc	
		X										X			write
															wait

inc dog



bus → IR	IR adr → bus	result → bus	bus → ALU	bus → oprnd	bus → PC	PC → bus	bus → R1	R1 → bus	bus → R2	R2 → bus	bus → MAR	bus → MBR	MBR → bus	ALU fun	Mem func
	x										x				read
															wait
			x										x	inc	
		x										x			write
															wait

Microcode Beyond the Simple Computer

- Next instruction field
  - Normally points to the next line
  - May point to fetch of next instruction
- ALU function taken from instruction register opcode
- Register selection from the instruction
- Status flags or comparison mechanism

Microphobia

An unreasoning fear of microcode

Fetch and PC Increment

- The fetch/execute cycle always starts with reading the instruction from memory and incrementing the program counter
- In our simple computer, this takes three lines of microcode
- This is the only time anything is loaded into the instruction register

Jump Instructions

- The last microcode step of a jump is almost always copies a value into the program counter
- Jump instructions rarely access memory unless they are pushing or popping something on the stack

Arithmetic

- Most arithmetic instructions have the steps
  - Copy something into the operand reg
  - Put a value on the bus and do it
  - Copy the result register someplace
- In a more realistic system, the ALU function would be determined by the opcode of the instruction

2nd Exam

- The next exam in COMP375 will be on Friday, October 25, 2019