Tutorial 7
CPUSim for A3

COMP2120B Computer organization

Kevin Lam (yklam2)

Overview

- CPUSim
- A3

CPUSim

- Free download at: http://www.cs.colby.edu/djskrien/CPUSim/
- A CPU simulator
- We can customize the simulation
- For the assignment, you **must** load and use comp2120.cpu

Running program in CPUSim

Assembly program

- A program consists of a sequence of machine instructions

Machine instructions

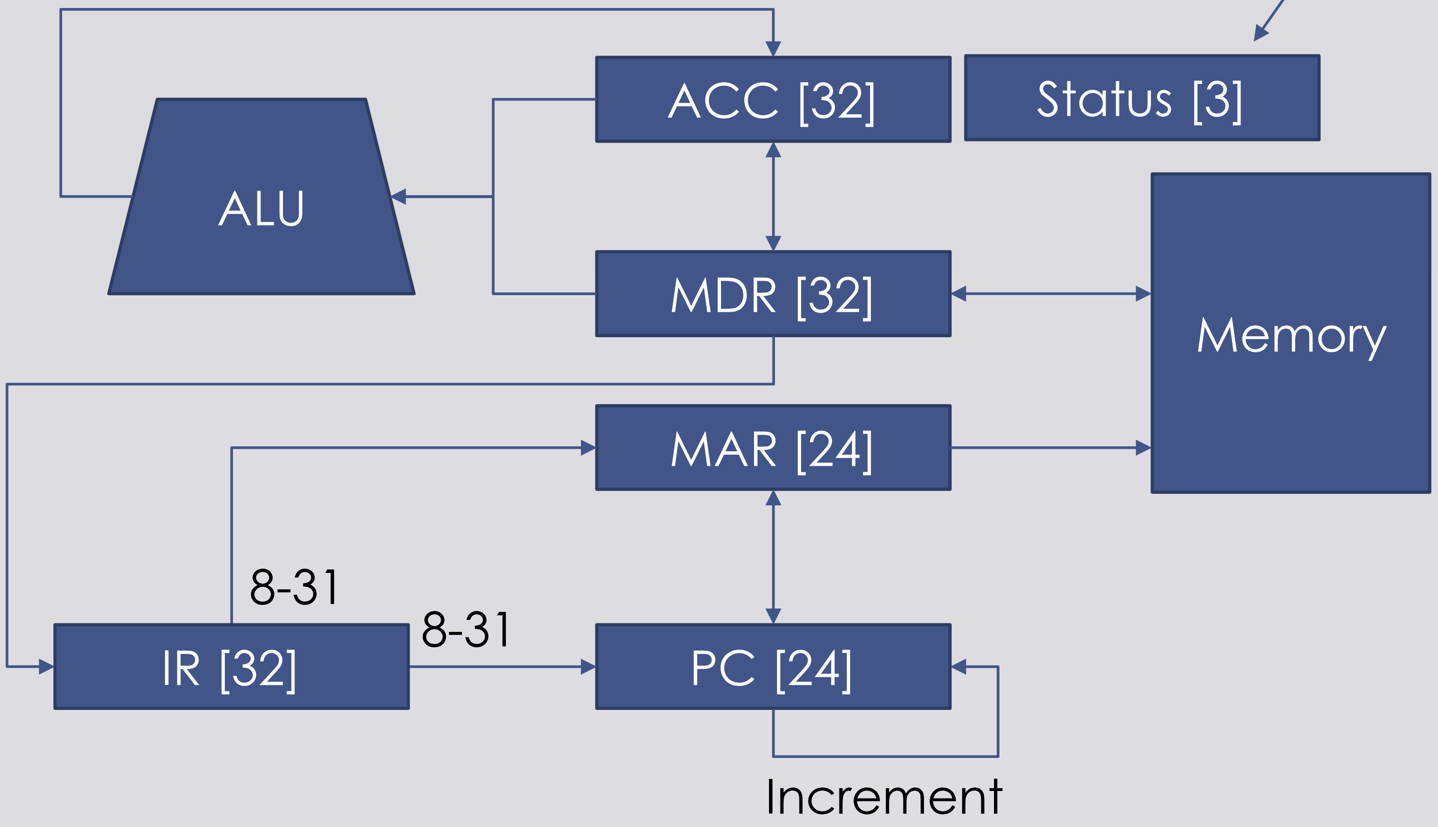
- An instruction consists of a sequence of micro-instructions

Micro-instructions

- A micro-instruction control how data is transmitted in the CPU

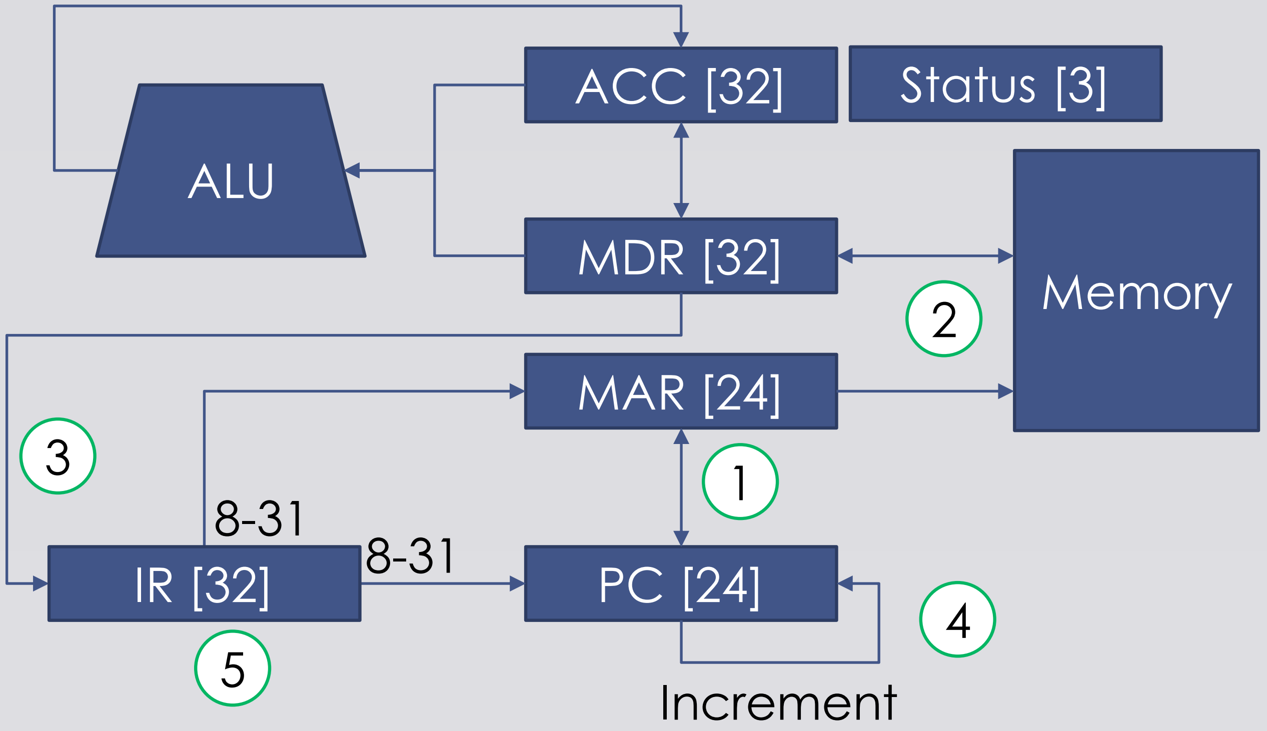
Architecture

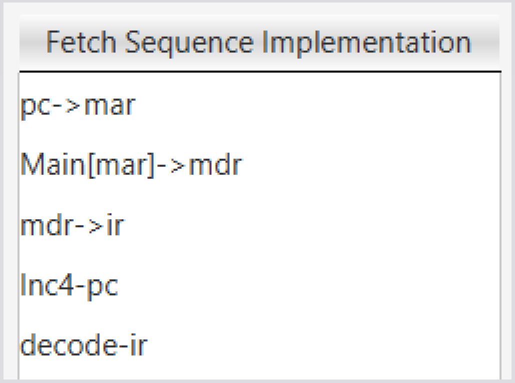
There is only one flag (HALT) in the **status**, which stop the execution of a program



Fetch sequence

- Fetch sequence is executed for every machine instruction
  - can be modified in Modify->Fetch Sequence

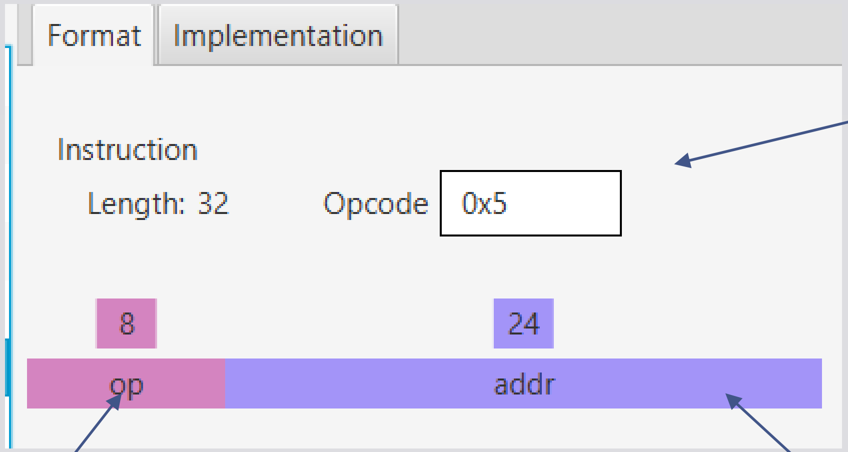



Fetch Sequence Implementation

```
pc->mar
Main[mar]->mdr
mdr->ir
Inc4-pc
decode-ir
```

The ADD instruction

- Modify → Machine Instructions → add



Instruction

Length: 32

Opcode 0x5

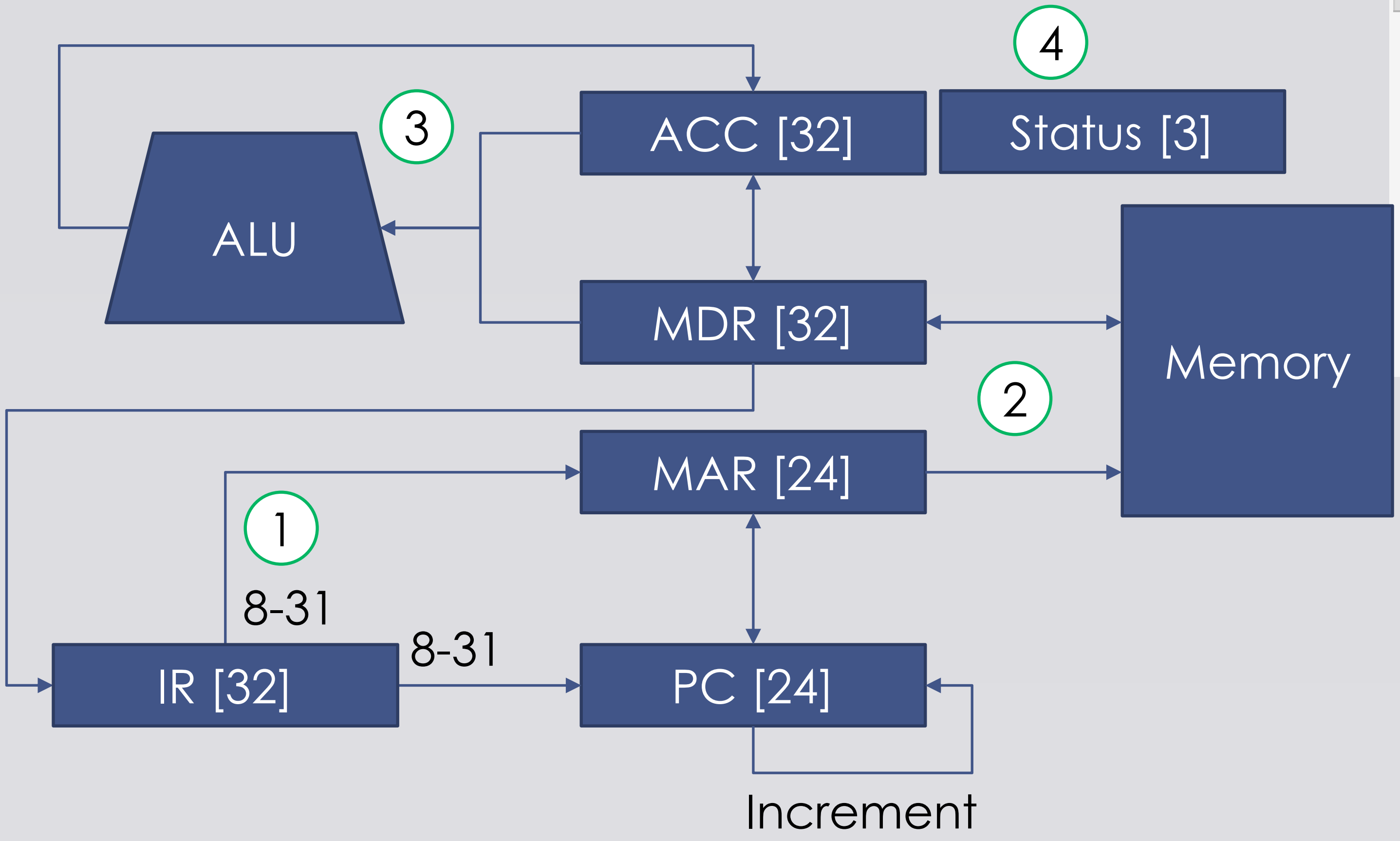
OP code is used to identify this instruction

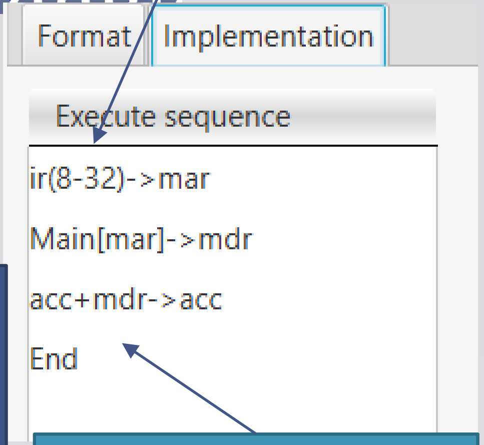
OP code takes 8 bits (bit 0-7)

Address takes 24 bits (bit 8-31)

Example instruction - ADD

Address takes 24 bits (bit 8-31)



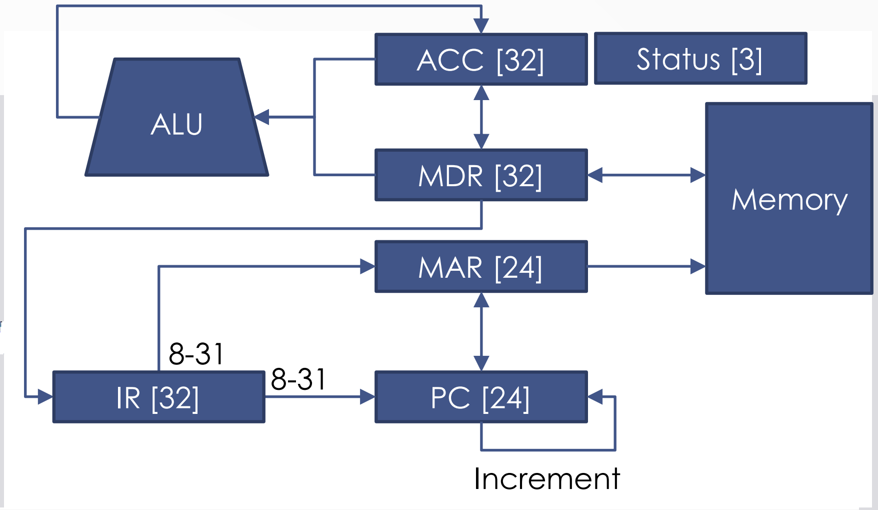


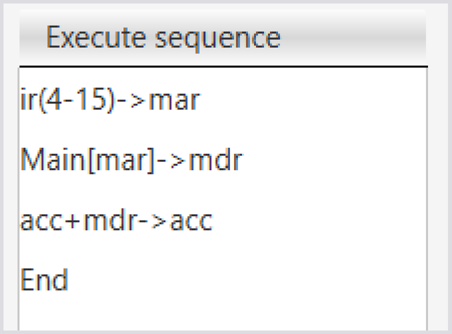
```
ir(8-32)->mar
Main[mar]->mdr
acc+mdr->acc
End
```

End instruction set up the halt bit in Status to end the program

Add instruction – micro-instructions

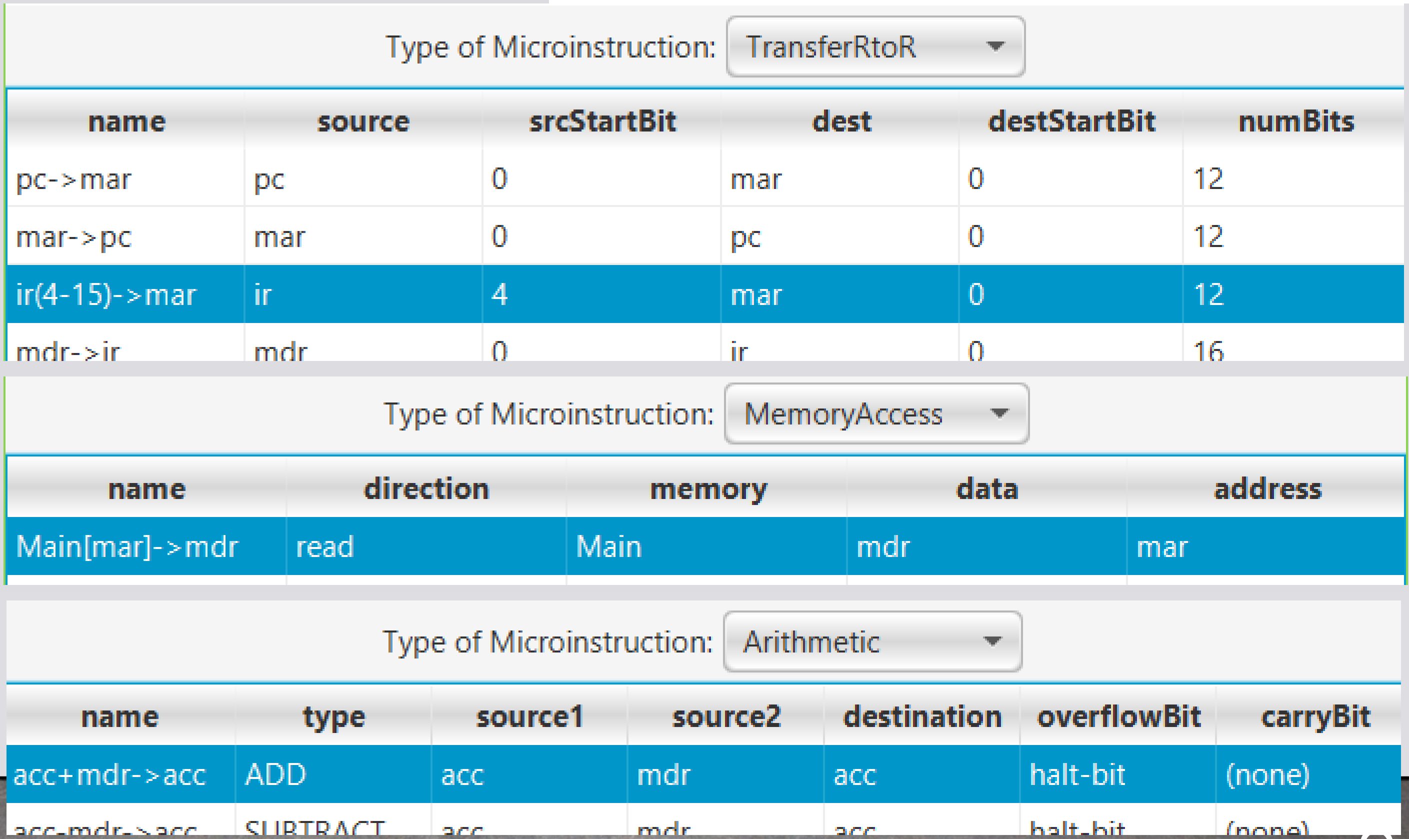
- Modify → Microinstructions





Execute sequence

ir(4-15)->mar
Main[mar]->mdr
acc+mdr->acc
End



Type of Microinstruction: TransferRtoR

name	source	srcStartBit	dest	destStartBit	numBits
pc->mar	pc	0	mar	0	12
mar->pc	mar	0	pc	0	12
ir(4-15)->mar	ir	4	mar	0	12
mdr->ir	mdr	0	ir	0	16

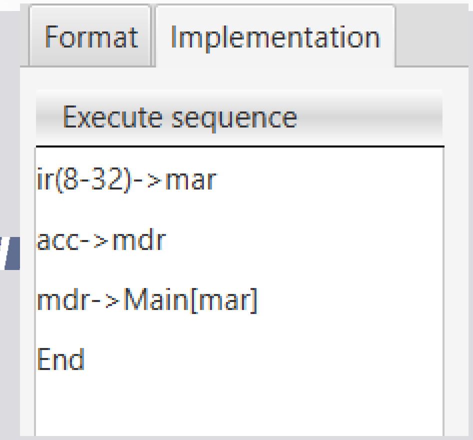
Type of Microinstruction: MemoryAccess

name	direction	memory	data	address
Main[mar]->mdr	read	Main	mdr	mar

Type of Microinstruction: Arithmetic

name	type	source1	source2	destination	overflowBit	carryBit
acc+mdr->acc	ADD	acc	mdr	acc	halt-bit	(none)
acc-mdr->acc	SUBTRACT	acc	mdr	acc	halt-bit	(none)

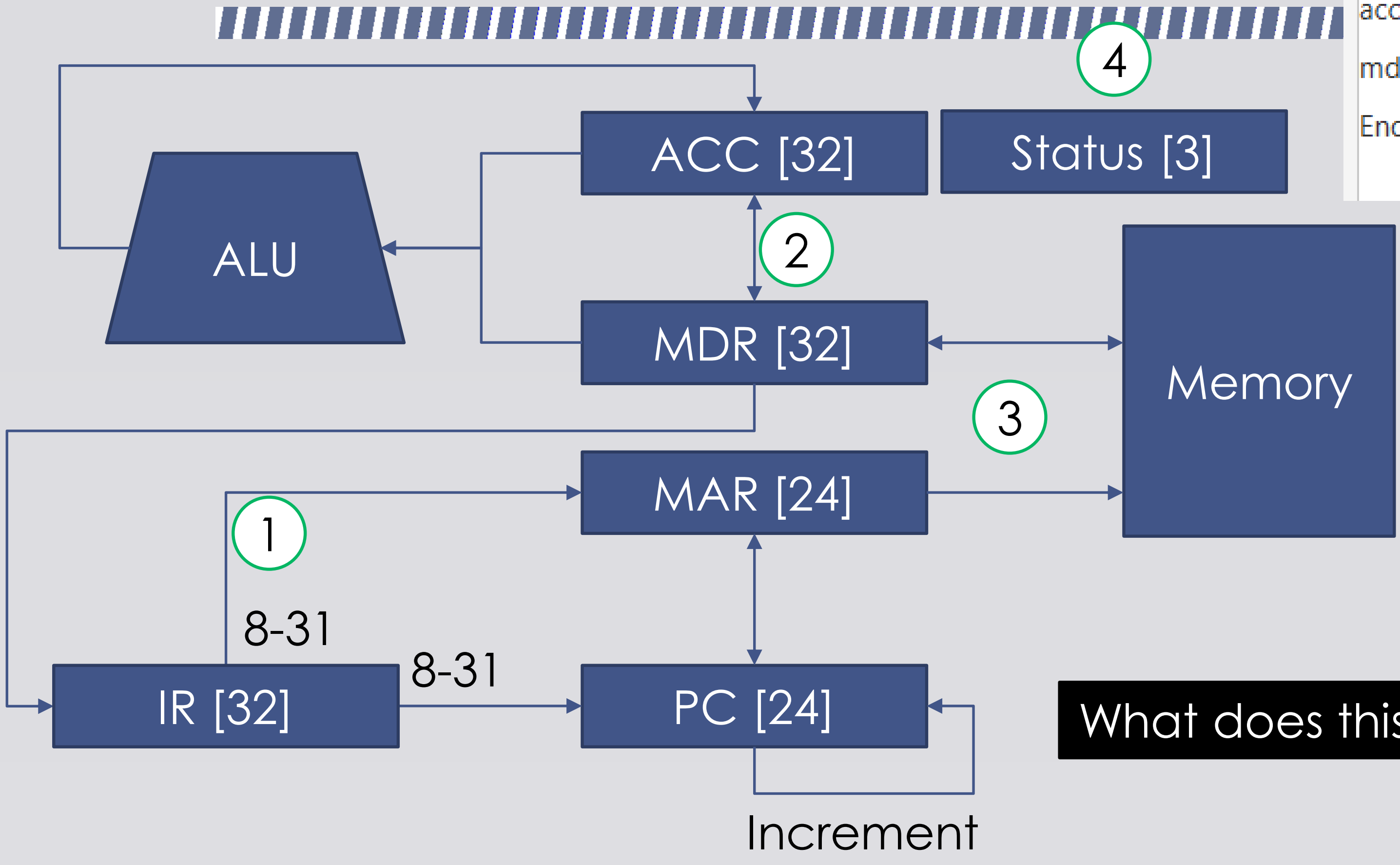
Another example - store



Format | Implementation

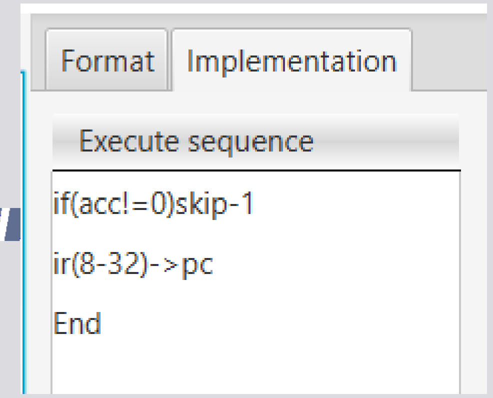
Execute sequence

```
ir(8-32)->mar
acc->mdr
mdr->Main[mar]
End
```

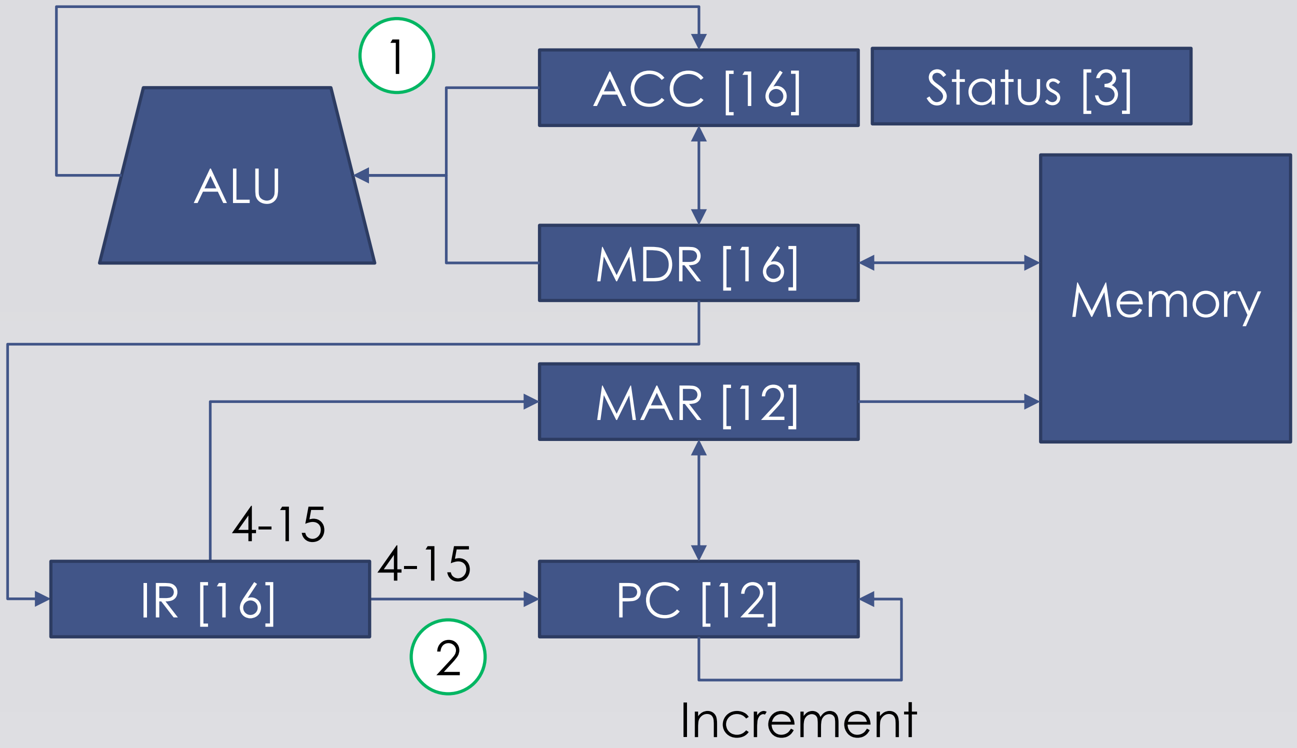


What does this instruction do?

Testing (for branching) - jmpz



```
if(acc!=0)skip-1
ir(8-32)->pc
End
```

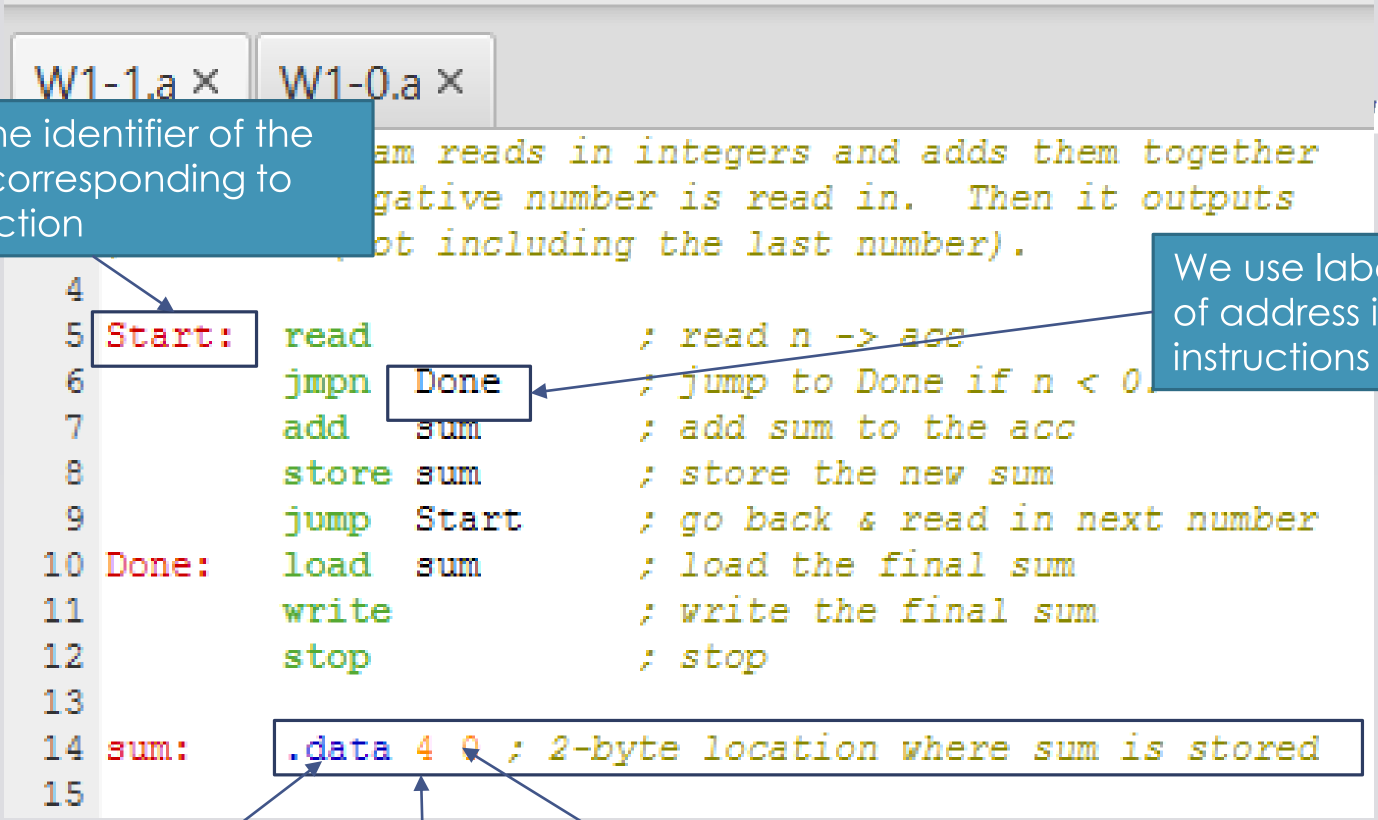


How to use it?

Instructions available

Instruction	Description
stop	End program
load <addr>	Find them out yourselves
store <addr>	
read	read input to ACC
write	print ACC to output
add <addr>	Find them out yourselves
subtract <addr>	
multiply <addr>	
divide <addr>	
jump <addr>	
jmpz <addr>	
jmpn <adr>	

Writing assembly program



```
            am reads in integers and adds them together
            gative number is read in.  Then it outputs
           (ot including the last number).

Start:  read            ; read n -> acc
        jmpn   Done     ; jump to Done if n < 0.
        add    sum      ; add sum to the acc
        store  sum      ; store the new sum
        jump   Start    ; go back & read in next number
Done:   load   sum      ; load the final sum
        write           ; write the final sum
        stop            ; stop

sum:    .data 4 0 ; 2-byte location where sum is stored
```

Label is the identifier of the address corresponding to the instruction

We use label instead of address in instructions

Define a data

Initialize to 0

2 bytes

Assignment 3

- There are 9 tasks.
- You can only use the microinstructions and machine instructions given by us plus those added in the assignment.