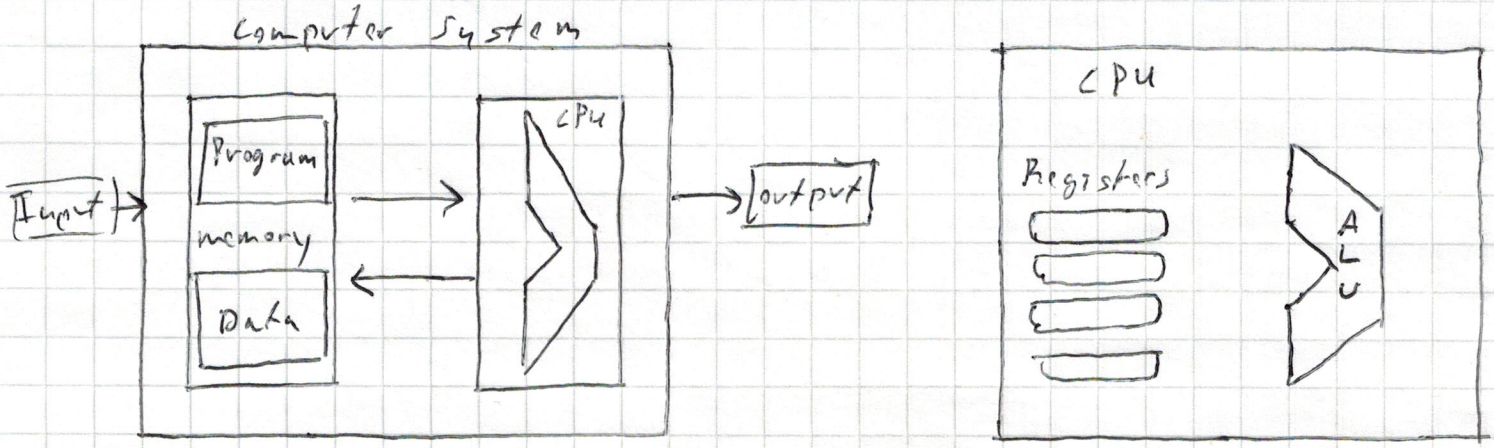
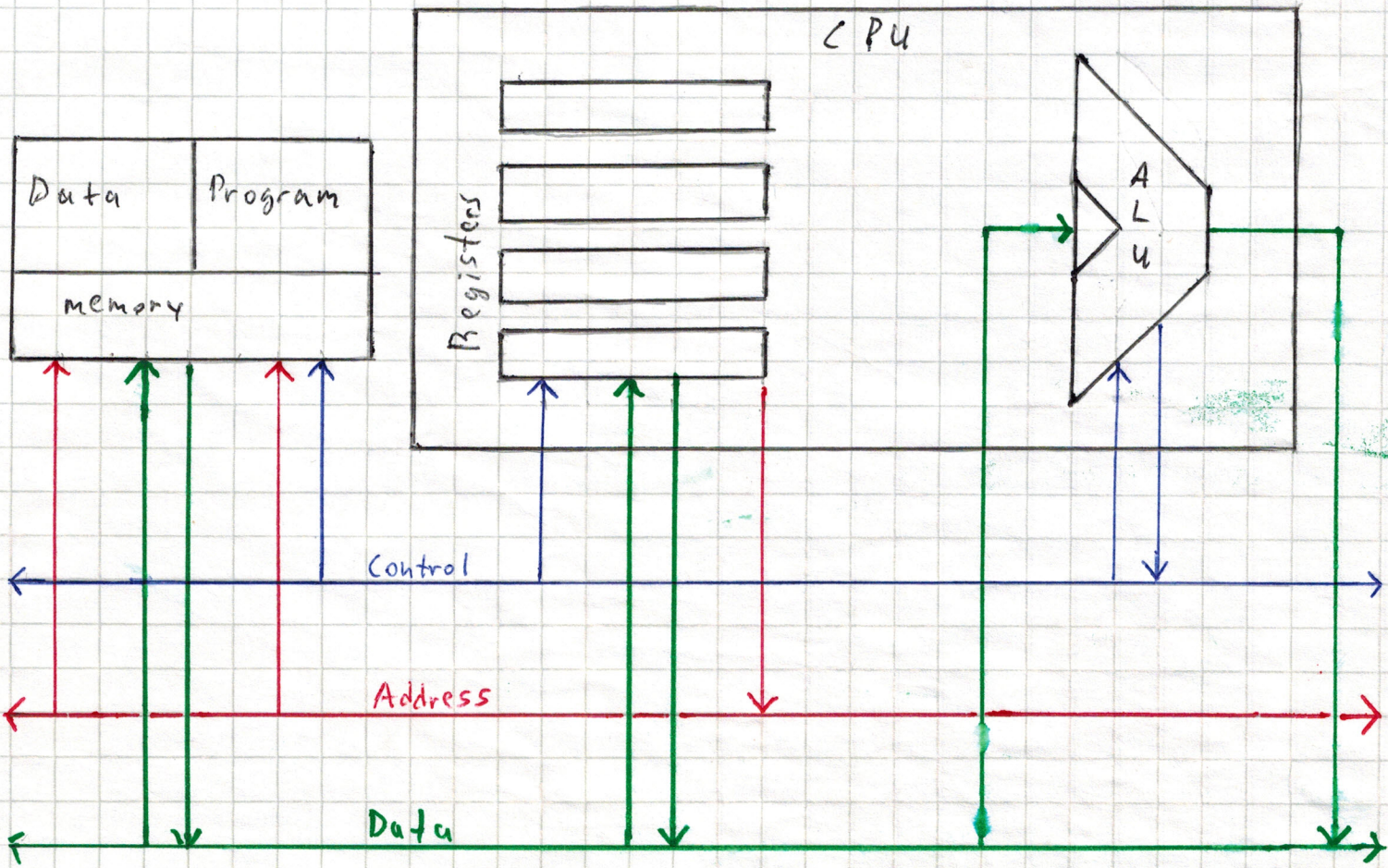
June 3, 2020

Unit 5.1 - Von Neumann Architecture



Information Flows

- 3 types:
  - Data (between memory, registers & ALU) - via data bus
  - Address (between memory & registers) - via address bus
  - Control (between memory, registers & ALU) - via control bits



June 8, 2020

Unit 5.2 – Fetch Execute Cycle

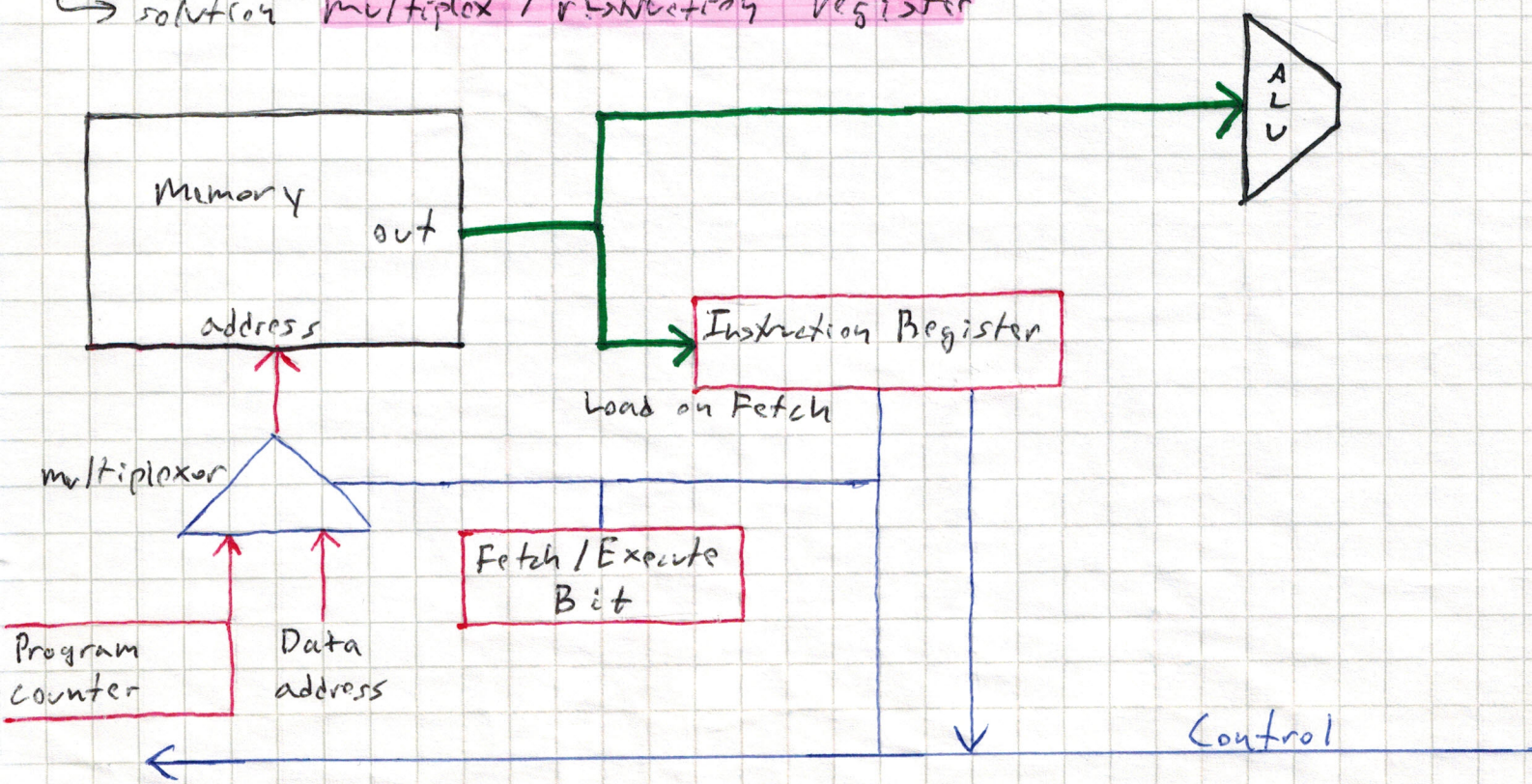
- The basic CPU loop:
  - Fetch an Instruction from program memory
  - Execute it

Fetching

- Put the location of the next instruction into "address" of program memory
- Get instruction code itself by reading the memory contents at that location

Executing

- The instruction code specifies "what to do"
  - which arithmetic or logical instruction
  - what memory to access (read/write)
  - If/where to jump
- often, different subsets of bits control different aspects of the operation – see **machine language spec.** – unit 4
- Executing the operation involves also accessing registers and/or data memory – so **Fetch–Execute clash** results (two memory to access?)
  ↳ solution **multiplex / instruction register**

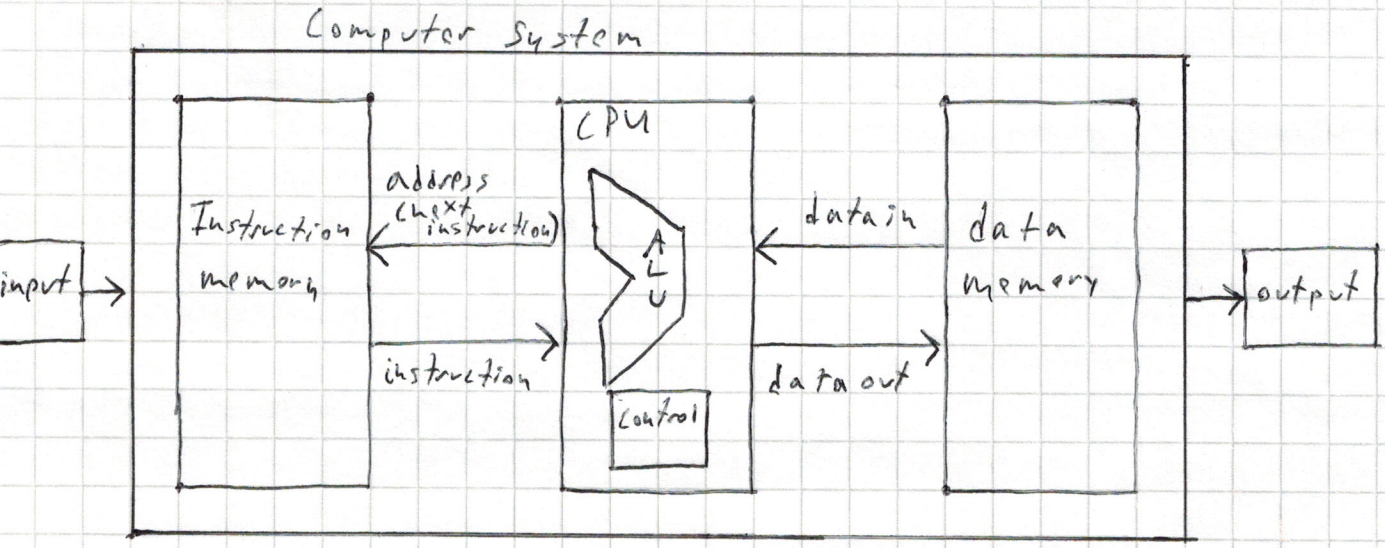


June 8, 2020

Unit 5.2 - Fetch-Execute Cycle (continued)

Simpler solution: Harvard Architecture
- Variant of Von Neumann Architecture
- Keep program & data in two separate memory modules
- Avoid complications

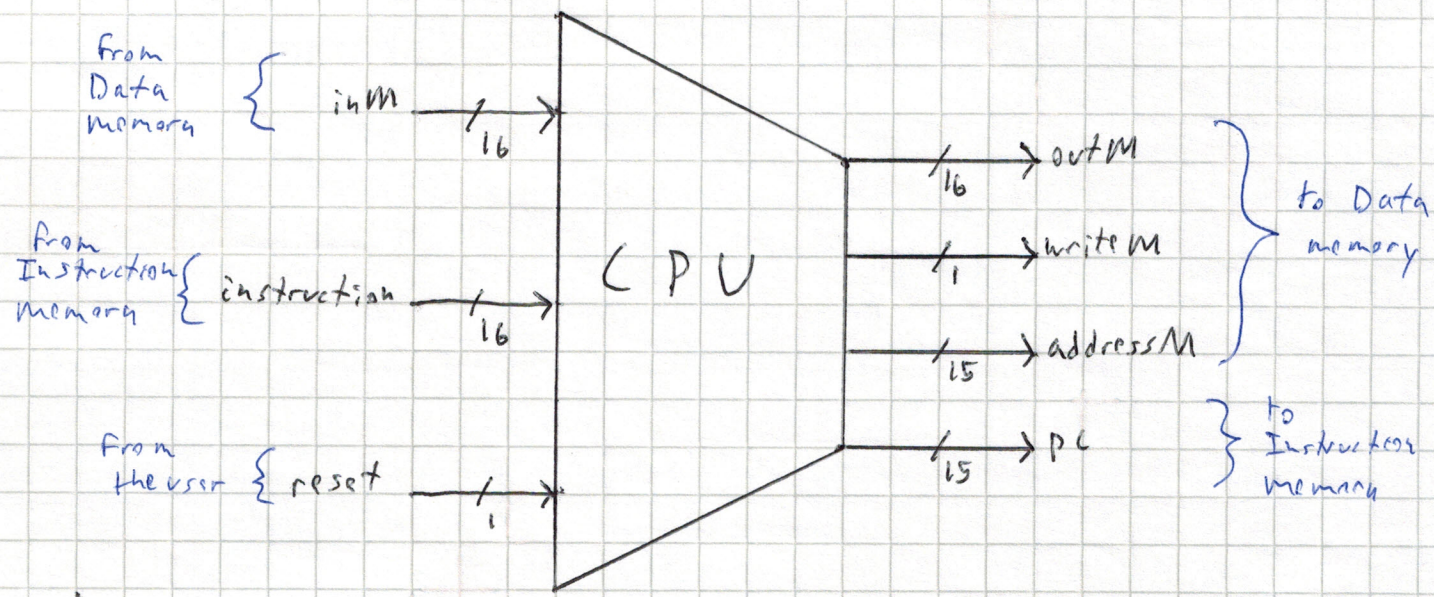
Unit 5.3 - Central Processing Unit



The Hack CPU: Abstraction

A 16-bit processor, designed to:
- Execute the current instruction (written in Hack language)
- Figure out which instruction to execute next

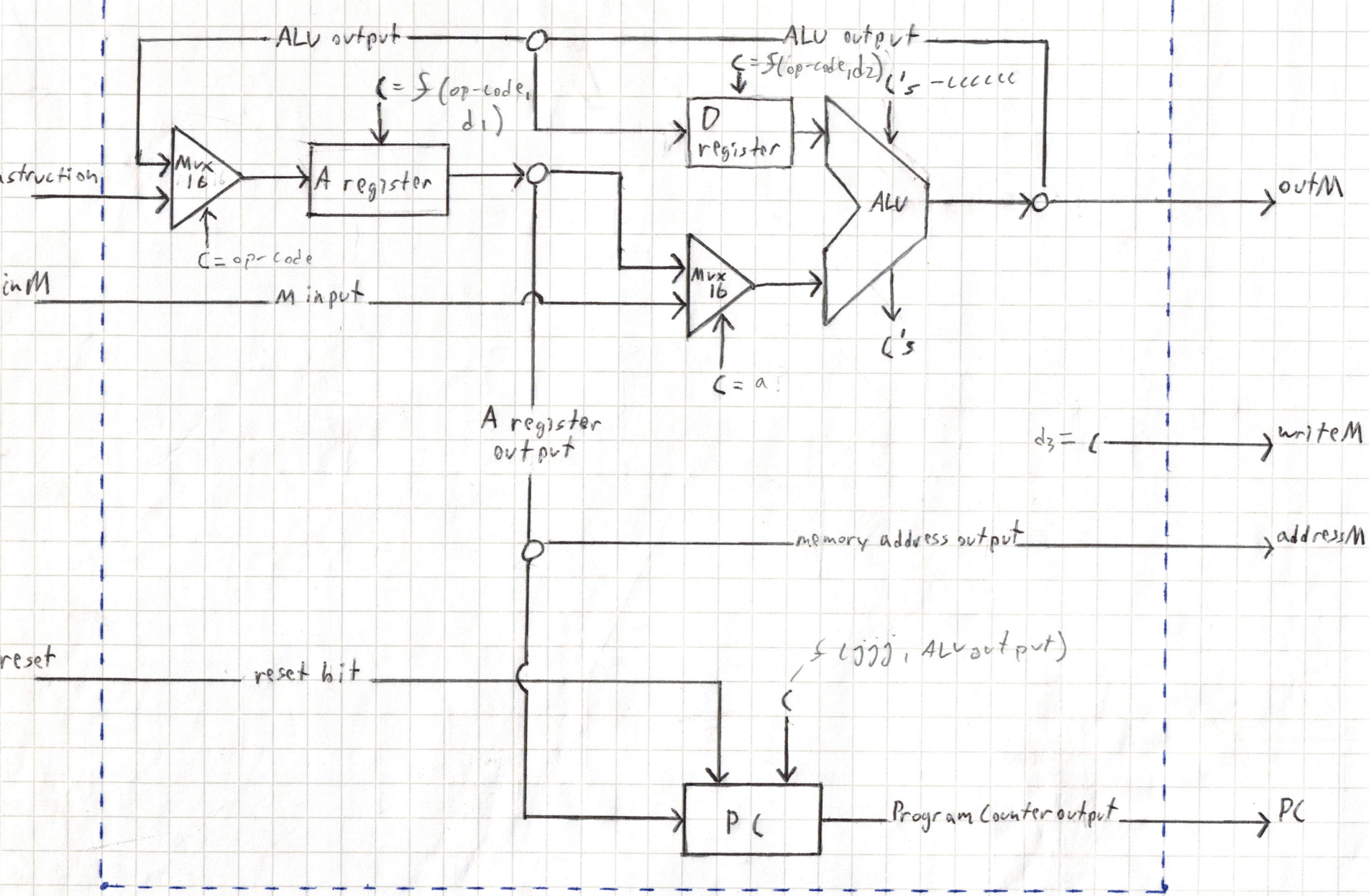
Hack CPU Interface



Inputs:
- Data value
- Instruction
- Reset bit

Outputs:
- Data value
- Write to memory? (yes/no)
- Memory Address
- Address of next instruction

Unit 5.3 - Central Processing Unit (continued)



C = control bits

recall { A-instruction: 0 ooooooooooooooo
C-instruction: 1 xx accccc ddd jjj

op-codes

Hack CPU Implementation

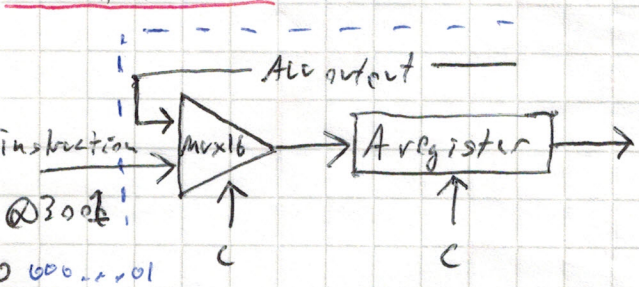
Dec 14, 2020

June 19, 2020

Unit 5.3 - Central Processing Unit (continued)

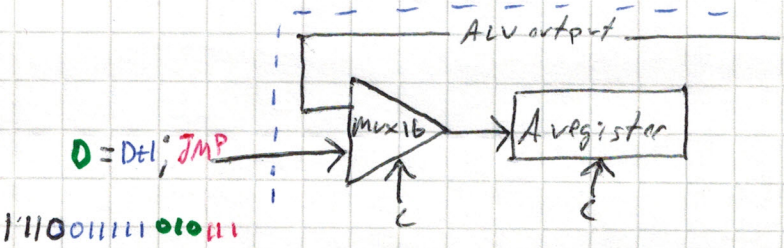
Instruction Handling

A-Instruction



- Decodes instruction op-code (1s-bit value
- Stores value in A-register
- Outputs value (address M output)

C-Instruction



Instruction bits decoded into:
- op-code = 1
- ALU control bits
- **Destination load bits**
- Jump bits

↳ ALU output gets routed into A-regis
↳ must inspect op-code

ALU Operation

Data Inputs:
- From D-register
- From A-register / M-register
  ↳ control bit for multiplexer taken from instruction

Control inputs
- Control bits from instruction (e.g. 011111)

Data output
- result of ALU calculation fed simultaneously to:
  - D-register, A-register, M-register
- Which register actually received incoming value is determined by instructions' **destination bits**
  ↳ they are routed to control bits of the registers

Control output
- Negative output?
- Zero output?

June 14, 2020

Unit 5.3 - Central Processing Unit (continued)

Reset Bit

- Computer is loaded with some program
- When you push reset, program starts running

CPU Control

recall C-instruction spec:

111a cccccc ddd jjj

Program Counter abstraction

Emits address of next instruction.

To start/restart the program's execution: PC = 0

- no jump: PC++
- goto: PC = A
- conditional goto: if condition true PC = A else PC++

Implementation

PC logic

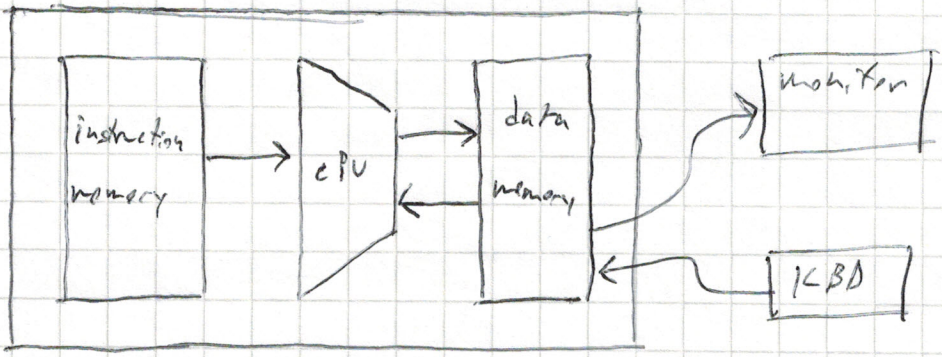
```
if (reset == 1) PC = 0
else
    // current instruction
    load = f(jump bits, ALU control outputs)
    if (load == 1) PC = A
    else           PC++
```

June 15, 2020

Unit 5.4 - The Hack Computer

Abstraction
- A computer capable of running programs written in the Hack machine language

Implementation
- Built from the Hack chip-set



Hack CPU Operation

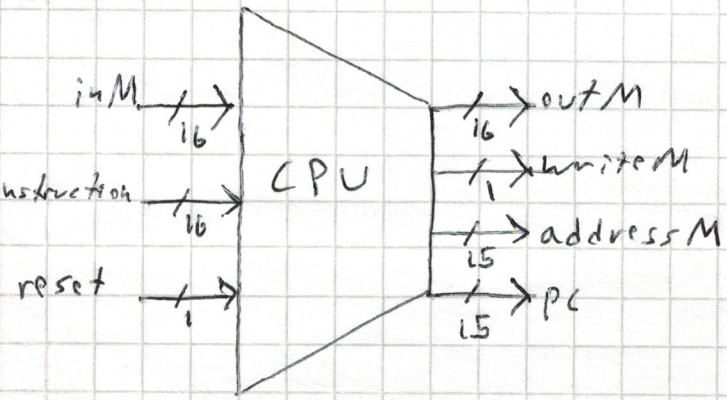
Sample Hack instructions:

D = D - A

@100
↳ D=D-1; JEQ

@17

M = M + 1



The CPU executes the instruction according to the Hack language specificati

- If the instruction includes D and A, the respective values are read from, and/or written to, the CPU resident D-register and A-registe
- If the instruction is @X, then X is stored in the A-register; this value is emitted by addressM
- If the instruction's BITS includes M, this value is read from inM
- If the instruction's LHS includes M, then the ALU output is emitted by outM, and the writeM bit is asserted
- If (reset == 0)
  - CPU logic uses instruction jump bits & ALU's output to decide if to jump
  - If there's a jump: PC is set to value of A-register
    Else (no jump): PC++
  - Updated PC value is emitted by pc
- If (reset == 1) → PC is set to 0, pc emits 0 (program restart)

June 13, 2020

Unit 5.4 - The Hack Computer (continued)

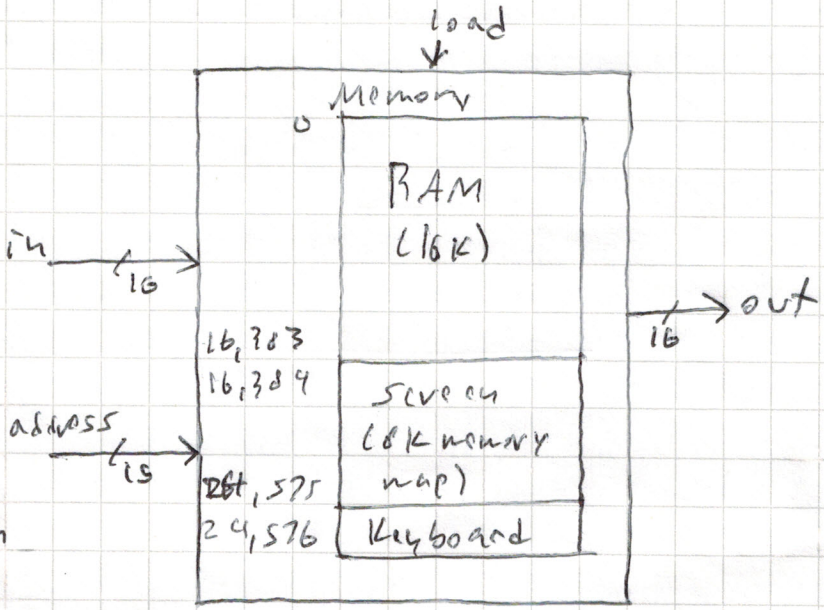
Data Memory

Implementation

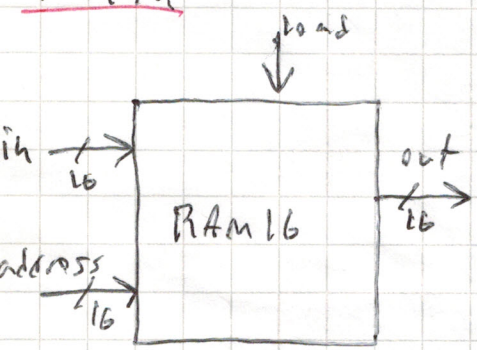
RAM: 16-bit/16K RAM chip

Screen: 16-bit/8K memory chip with raster display side effect

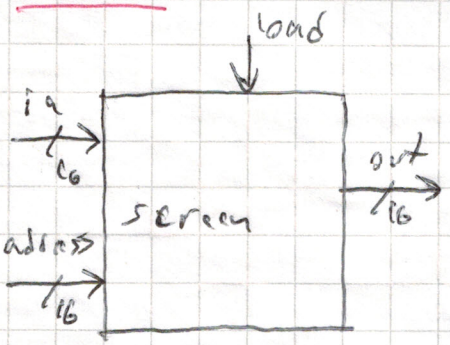
Keyboard: 16-bit register with a keyboard sideeffect



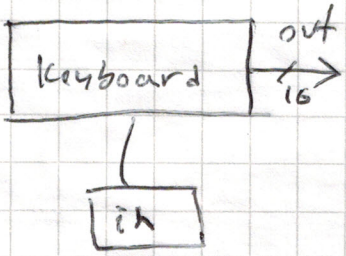
RAM



Screen



Keyboard



* note RAM16 / Screen memory chips have same interface as overall data memory chip

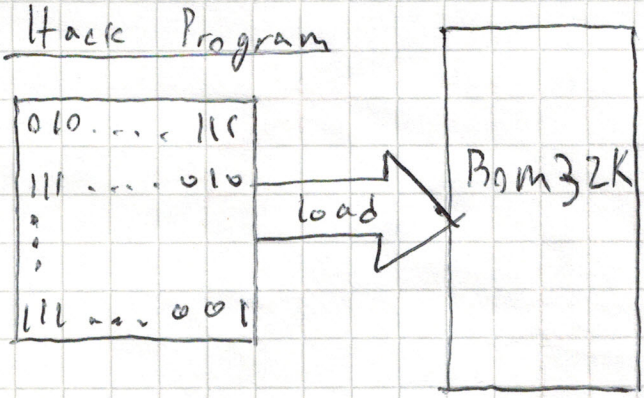
* refer to unit 4 for notes on I/O & memory maps

June 15, 2020

Unit 5.4 - The Hack Computer (continued)

Instruction Memory (ROM)

Hack Program



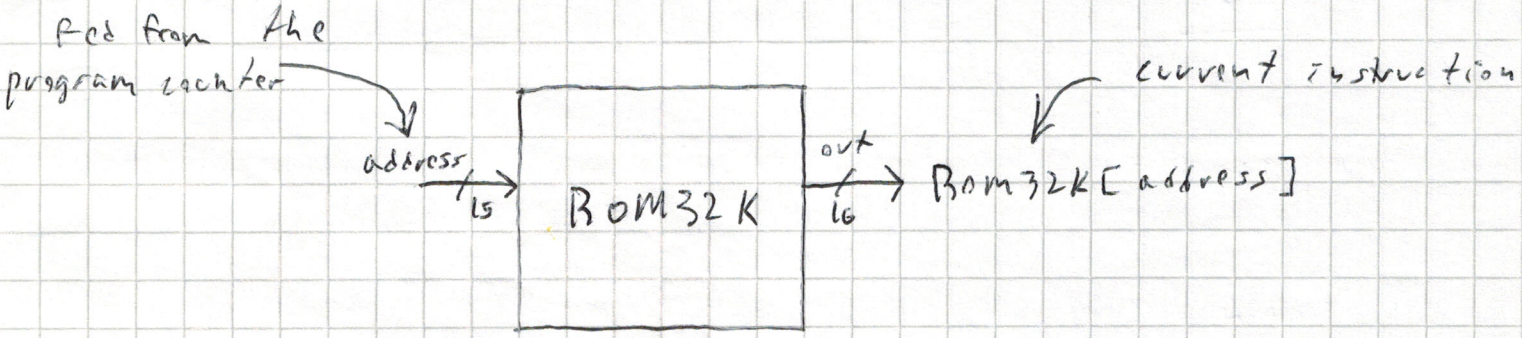
To run a program on Hack computer

- load program into Rom
- Press "reset"
- The program starts running

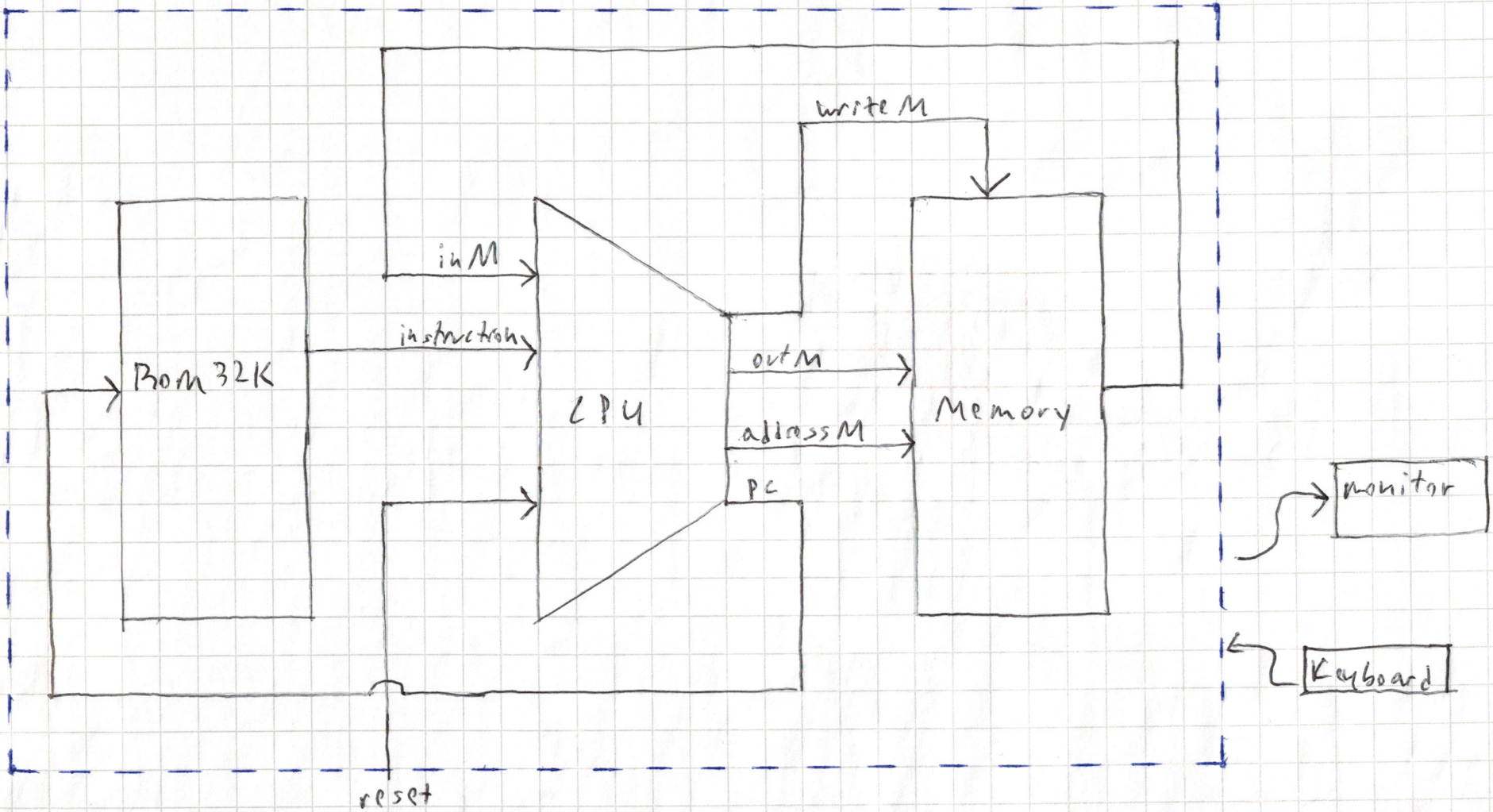
Loading a Program

- Hardware implementation: interchangeable, plug-and-play Rom chips
  e.g. video game cartridges
- Hardware simulation: programs are stored in text files; program loading is emulated by the built-in Rom chip

Rom Interface



- Implemented as a built-in Rom32K chip
- Contains a sequence of Hack instructions (program)



Hack Computer Implementation

Unit 5 - Program Counter Control in Hack CPU

c = f(jjj, zr, ng)

jjj = 3 jump bits from Hack Machine language C-instruction

zr, ng = control output bits of ALU chip

	j1 (out<0)	j2 (out=0)	j3 (out>0)	Mnemonic	Effect
	0	0	0	null	No jump
	0	0	1	JGT	If out>0 jump
	0	1	0	JEQ	If out=0 jump
	0	1	1	JGE	If out ≥0 jump
	1	0	0	JLT	If out <0 jump
	1	0	1	JNE	If out ≠0 jump
	1	1	0	JLE	If out ≤0 jump
	1	1	1	JMP	Jump
zr	0	1	0		
ng	1	0	0		

reference: figure 4.5

Load = (j1 AND ng OR j2 AND zr OR j3 AND NOT ng AND NOT zr) AND (opcode)