CPSC 121: Models of Computation

Unit 10: A Working Computer

Based on slides by Patrice Belleville

Learning Goals

- After completing Lab 9 and this unit, you should be able to:
  - Specify the overall architecture of a (Von Neumann) stored program computer - an architecture where both program and data are bits (i.e., state) loaded and stored in a common memory.
  - Trace execution of an instruction through a working computer in a logic simulator (currently logisim): the basic fetch-decode-execute instruction cycle and the data flow to/from the arithmetic logic unit (ALU), the main memory and the Program Counter (PC).
  - Feel confident that, given sufficient time, you could understand how the circuit executes machine-language instructions.

CPSC 121 Big Questions

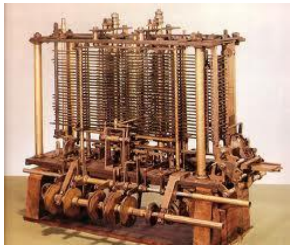
- CPSC 121: the BIG questions:
  - How can we build a computer that is able to execute a user-defined program?
- We are finally able to answer this question.
  - This unit summarizes the concepts related to hardware you've learned in the lectures and labs since the beginning of the term.

Outline

- **A little bit of history**
- Implementing a working computer in Logisim
- Appendices

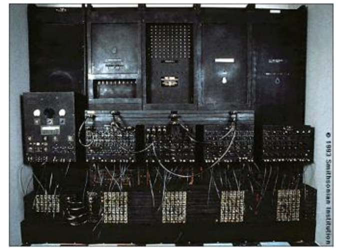
Computer History

- Early 19th century:
  - Joseph Marie Charles dit Jacquard used punched paper cards to program looms.
  - Charles Babbage designed (1837) but could not build the first programmable (mechanical) computer, based on Jacquard's idea.
  - Difference Engine 2 built in London in 2002
    - 8000 parts
    - 11 feet long
    - 5 tons



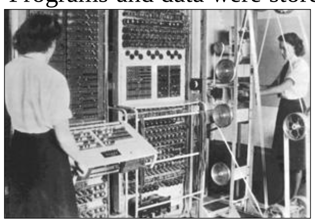
Computer History (cont')

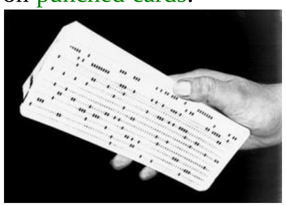
- 20th century
  - Konrad Zuse (1941) build the first electromechanical computer (Z3). It had binary arithmetic (including floating point) and was programmable.
  - The ENIAC (1946) was the first programmable electronic computer.
    - It used decimal arithmetic.
    - Reprogramming it meant rewiring it!



Computer History (cont')

- Mid 20th century:
  - The first stored-program electronic computers were developed from 1945 to 1950.
  - Programs and data were stored on punched cards.





- More on http://www.computerhistory.org

Computer Architecture Related Courses

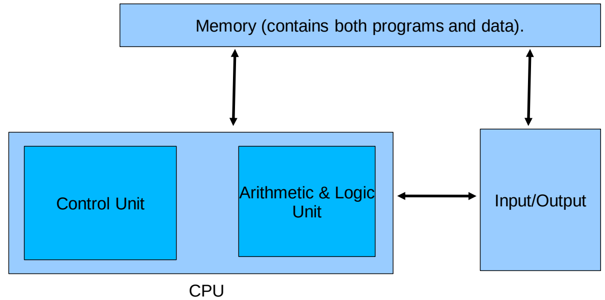
- A quick roadmap through our courses:
  - CPSC 121: learn about gates, and how we can use them to design a circuit that executes very simple instructions.
  - CPSC 213: learn how the constructs available in languages such as Racket, C, C++ or Java are implemented using simple machine instructions.
  - CPSC 313: learn how we can design computers that execute programs efficiently and meet the needs of modern operating systems.

Outline

- A little bit of history
- **Implementing a working computer in Logisim**
- Appendices

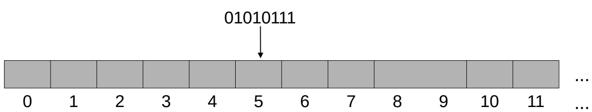
Modern Computer Architecture

- First proposed by Von-Neumann in 1945.



Memory

- Contains both instructions and data.
- Divided into a number of memory locations
  - Think of positions in a list: (list-ref mylist pos)
  - Or in an array: myarray[pos] or arrayList arrayl.get(pos).



Memory (cont')

- Each memory location contains a fixed number of bits.
  - Most commonly this number is 8.
  - Values that use more than 8 bits are stored in multiple consecutive memory locations.
    - Characters use 8 bits (ASCII) or 16/32 (Unicode).
    - Integers use 32 or 64 bits.
    - Floating point numbers use 32, 64 or 80 bits.

Central Processing Unit (CPU)

- Arithmetic and Logic Unit
  - Performs arithmetic and logical operations (+, -, *, /, and, or, etc).
- Control Unit
  - Decides which instructions to execute.
  - Executes these instructions sequentially.
    - Not quite true, but this is how it appears to the user.

Our Working Computer

- Implements the design presented in the textbook by Bryant and O'Hallaron (used for CPSC 213/313).
- A small subset of the IA32 (Intel 32-bit) architecture.
- It has
  - 12 types of instructions.
  - One program counter register (PC)
    - contains the address of the next instruction.
  - 8 general-purpose 32-bits registers
    - each of them contains one 32 bit value.
    - used for values that we are currently working with.

stores a single multi-bit value.

Instruction Examples

instruction register

- Example instruction 1: subl %eax, %ebx
  - The subl instruction subtracts its arguments.
  - The names %eax and %ebx refer to two registers.
  - This instruction takes the value contained in %eax, subtracts it from the value contained in %ebx, and stores the result back in %ebx.

instruction constant register

- Example instruction 2: irmovl $0x1A, %ecx
  - This instruction stores a constant in a register.
  - In this case, the value 1A (hexadecimal) is stored in %ecx.

Instruction Examples (cont')

instruction register memory location

- Example instruction 3: rmmovl %ecx, $8(%ebx)
  - The rmmovl instruction stores a value into memory (Register to Memory Move).
  - In this case it takes the value in register %ecx.
  - And stores it in the memory location whose address is:
    - The constant 8
    - PLUS the current value of register %ebx.

Instruction Examples (cont')

- Example instruction 4: jge $1000
  - This is a conditional jump instruction.
  - It checks to see if the result of the last arithmetic or logic operation was zero or positive (Greater than or Equal to 0).
  - If so, the next instruction is the instruction stored in memory address 1000 (hexadecimal).
  - If not, the next instruction is the instruction that follows the jge instruction.

Sample program:

```
irmovl $3,%eax
irmovl $35, %ebx
irmovl $facade, %ecx
subl %eax, %ebx
rmmovl %ecx, $8(%ebx)
halt
```


Instruction Format

- How does the computer know which instruction does what?
  - Each instruction is a sequence of 16 to 48 bits†
  - Some of the bits tell it what type of instruction it is.
  - Other bits tell it which instruction is and what operands to use.
- These bits are used as control (select) inputs for several multiplexers.

† Modified slightly from the Y86 presented in the textbook by Bryant and O'Hallaron

Instruction Examples

- Example 1: subl %eax, %ebx
  - Represented by
    - 6103 (hexadecimal)

      %ebx

      %eax

      subtraction

      arithmetic or logic operation

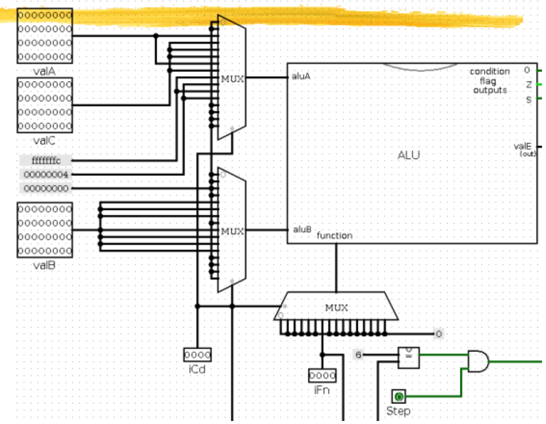
      (note: the use of "6" to represent them instead of 0 or F or any other value is completely arbitrary)..

Instruction Examples (cont')

- Example 2: rmmovl %ecx, $8(%ebx)
  - Represented by
    - 401300000008 (hexadecimal)
      - $8
      - %ebx
      - %ecx
      - ignored
      - register to memory move

A Working Computer in Logisim

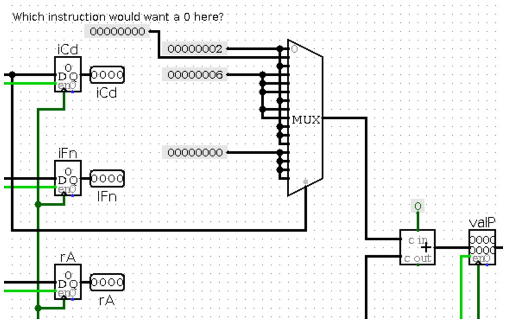
- Example:



Instruction Execution Stages

This CPU divides the instuction execution into 6 stages:

- **Fetch**: read instruction and decide on new PC value



Instruction Execution Stages (cont')

- **Decode**: read values from registers
- **Execute**: use the ALU to perform computations
  - Some of them are obvious from the instruction (e.g. subl)
  - Other instructions use the ALU as well (e.g. rmmovl)
- **Memory**: read data from or write data to memory
- **Write-back**: store result(s) into register(s).
- **PC update**: store the new PC value.
- Not all stages do something for every instruction.

Sample Program

irmovl $3,%eax	30f000000003
irmovl $35, %ebx	30f300000023
irmovl $facade, %ecx	30f100facade
subl %eax, %ebx	6103
rmmovl %ecx, $8(%ebx)	411300000008
halt	1000

Instruction Execution Examples

- Example 1: subl %eax, %ebx
  - **Fetch**: current instruction ← 6103
  - next PC value ← current PC value + 2
  - **Decode**: valA ← value of %eax
  - valB ← value of %ebx
  - **Execute**: valE ← valB - valA
  - **Memory**: nothing needs to be done.
  - **Write-back**: %ebx ← valE
  - **PC update**: PC ← next PC value

nstruction Execution Examples (cont')

- Example 2: rmmovl %ecx, $8(%ebx)
  - **Fetch**: current instruction ← 401300000008
  - next PC value ← current PC value + 6
  - **Decode**: valA ← value of %ecx
  - valB ← value of %ebx
  - **Execute**: valE ← valB + 00000008
  - **Memory**: M[valE] ← valA
  - **Write-back**: nothing needs to be done
  - **PC update**: PC ← next PC value

Outline

- A little bit of history
- Implementing a working computer in Logisim
- **Appendices**

Appendix 1: Registers and Memory

- Registers (32 bits each):

%eax	0	%esp	4
%ecx	1	%ebp	5
%edx	2	%esi	6
%ebx	3	%edi	7

  - Instructions that only need one register use 8 or F for the second register.
  - %esp is used as stack pointer.
- Memory contains 2^{32} bytes; all memory accesses load/store 32 bit words.

Appendix 2: Instruction Types

- Register/memory transfers:
  - rmmovl rA, D(rB) M[D + R[rB]] ← R[rA]
    - Example: rmmovl %edx, 20(%esi)
  - mrmovl D(rB), rA R[rA] ← M[D + R[rB]]
- Other data transfer instructions
  - rrmovl rA, rB R[rB] ← R[rA]
  - irmovl V, rB R[rB] ← V

Instruction Types (cont')

- Arithmetic instructions
  - addl rA, rB R[rB] ← R[rB] + R[rA]
  - subl rA, rB R[rB] ← R[rB] − R[rA]
  - andl rA, rB R[rB] ← R[rB] ∧ R[rA]
  - xorl rA, rB R[rB] ← R[rB] ⊕ R[rA]

Instruction Types (cont')

- Unconditional jumps
  - jmp Dest PC ← Dest
- Conditional jumps
  - jle Dest PC ← Dest if last result ≤ 0
  - jl Dest PC ← Dest if last result < 0
  - je Dest PC ← Dest if last result = 0
  - jne Dest PC ← Dest if last result ≠ 0
  - jge Dest PC ← Dest if last result ≥ 0
  - jg Dest PC ← Dest if last result > 0

Instruction Types (cont')

- Conditional moves
  - cmovle rA, rB — R[rB] ← R[rA] if last result ≤ 0
  - cmovl rA, rB — R[rB] ← R[rA] if last result < 0
  - cmove rA, rB — R[rB] ← R[rA] if last result = 0
  - cmovne rA, rB — R[rB] ← R[rA] if last result ≠ 0
  - cmovge rA, rB — R[rB] ← R[rA] if last result ≥ 0
  - cmovg rA, rB — R[rB] ← R[rA] if last result > 0

Instruction Types (cont')

- Procedure calls and return support
  - call Dest — R[%esp]←R[%esp]-4; M[R[%esp]]←PC; PC←Dest;
  - ret — PC←M[R[%esp]]; R[%esp]←R[%esp]+4
  - pushl rA — R[%esp]←R[%esp]-4; M[R[%esp]]←R[rA]
  - popl rA — R[rA]←M[R[%esp]]; R[%esp]←R[%esp]+4
- Others
  - halt — stop execution
  - nop — no operation

Appendix 3: Instruction Format

Instruction	0	1	2	3	4	5
nop	0 0	0 0				
halt	1 0	0 0				
cmovXX rA, rB	2 fn	rA rB				
irmovl V, rB	3 0	F rB	V			
rmmovl rA, D(rB)	4 0	rA rB	D			
mrmovl D(rB), rA	5 0	rA rB	D			
OPl rA, rB	6 fn	rA rB				
jXX Dest	7 fn	0 0	Dest			
call Dest	8 0	0 0	Dest			
ret	9 0	0 0				
pushl rA	A 0	rA F				
popl rA	B 0	rA F				

Instruction Format (cont')

- Instructions format:
  - Arithmetic instructions:
    - addl → fn = 0 subl → fn = 1
    - andl → fn = 2 xorl → fn = 3
  - Conditional jumps and moves:
    - jump → fn = 0 jle → fn = 1
    - jl → fn = 2 je → fn = 3
    - jne → fn = 4 jge → fn = 5
    - je → fn = 6