The von Neumann model

Lecture Topics

FSM (continued from last lecture)

Von Neumann model

LC-3 as a von Neumann machine

Lecture materials

Textbook Ch. 4

Textbook Appendix C

Homework

HW2 due February 17 at 5pm in the ECE 190 drop-off box

Machine problem

MP1.2 due February 17 at 5pm submitted electronically.

Announcements

Exam is next week. Check Compass for your day/room exam assignment.

Remember you can have one sheet of paper with hand-written notes at the exam. No books or calculators.

FSM (continued from last lecture)

Finish last lecture's FSM example

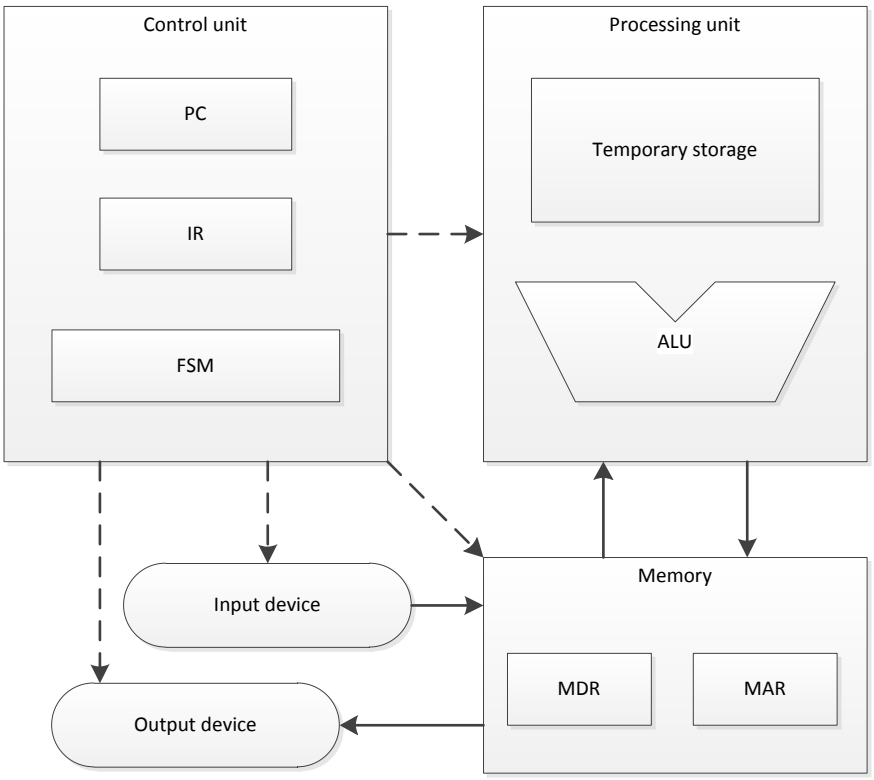
Von Neumann model

Concept

- The von Neumann model/architecture refers to a computer organization which is based on the *stored-program* concept
- Stored-program concept refers to the computer architecture in which data and program are stored in the same memory
    - The von Neumann architecture ≈ stored-program concept
- First computers built in 1940's were *program-controlled* computers
    - Programmed by setting switches and physically connecting functional units
- Stored-program digital computes kept their program (set of instructions) and data in read-write random-access memory

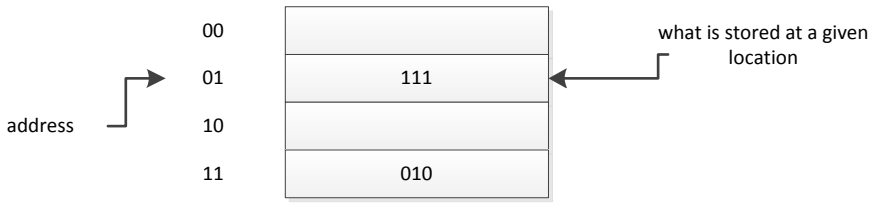
Von Neumann model components

- Memory – stores data and program
- Processing unit – performs the data processing
- Input – means to enter data and program
- Output – means to extract results
- Control unit – controls the order of the instruction execution

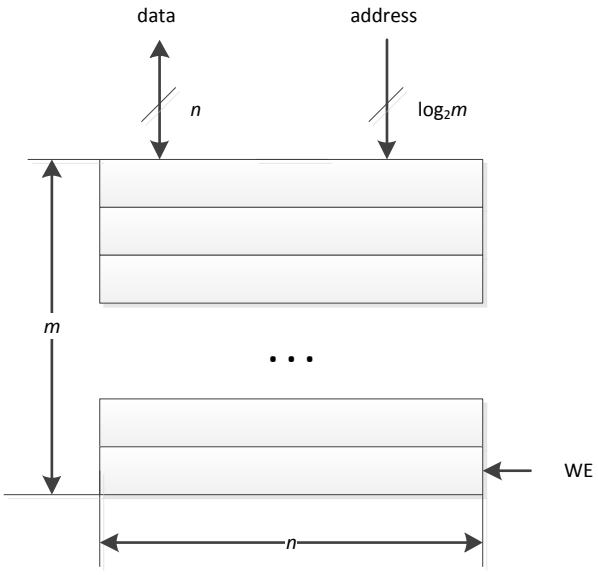


Memory

- In the last lecture we built a 2^2-by-3-bit memory
  - 3-bit addressability – at each memory location we can store 3 bits of information
  - 2^2 address space – there are that many unique memory locations that can be addressed individually
  - This memory can be thought of as an array, or sequence of locations that are sequentially numbered



    - note *address* and the *value stored in memory* at that address
  - In this example, memory location '01' contains value '111' and memory lication'11' contains value '010'
- A modern computer with 1 GB of memory
  - 8-bit (one byte) addressability – at each memory location we can store one byte of data
  - 2^{30} address space - 2^{30} uniquely addressable memory locations
- In general, we will think of memory as a device capable of storing *m* *n*-bit words
  - *n* – addressability
  - *m* – address space
  - to interact with the memory, we will need *n*-bit data line, $\log_2 m$-bit address line, and 1-bit WE line



- Actual interaction with memory is implemented using two special-purpose registers, called MAR (memory address register) and MDR (memory data register)
- To read a value form memory
    - Place the address of the memory location to read from in the MAR
        - Memory circuit then will transfer data from memory to MDR
    - Read value from MDR
- To write a value
    - Place the address of the memory location to write to in the MAR
    - Place the value to be stored in the memory in the MDR
    - Assert (set to 1) WE signal
        - Memory circuit then will transfer data from MDR to memory

Processing unit

- Carries out actual processing of information
- In the simples form, it consists of two main parts
    - An ALU that implements a few basic operations, such as ADD, AND, etc.
        - As an example, LC-3 computer's ALU has ADD, AND, and NOT operations only
    - A temporary storage, typically a set of few registers for storing few words of data
        - As an example, LC-3 computer has 8 registers
    - The size of data items processed by the ALU is referred to as the *word length*, and each data item is referred to as a *word*
        - As an example, LC-3 computer has word length of 16 bits (or two bytes)
        - In a modern 64-bit microprocessor, word length is 64 bits (or 8 bytes)

Input and Output

- In order for computer to process the information, the information itself needs to be entered into the memory
- In order for us to know the results of processing the information, the results need to be output in such a way that we can see them
- To accomplish this, computers have some form of input and output devices, generically referred to as *peripherals*
- As an example, LC-43 computer has keyboard as an input device and monitor as an output device
- In the simplest form, input and output devices work with memory directly, that is, an input device places a value into some memory location and an output device reads and displays a value from some memory location
    - Such I/O is referred to as *memory-mapped* I/O

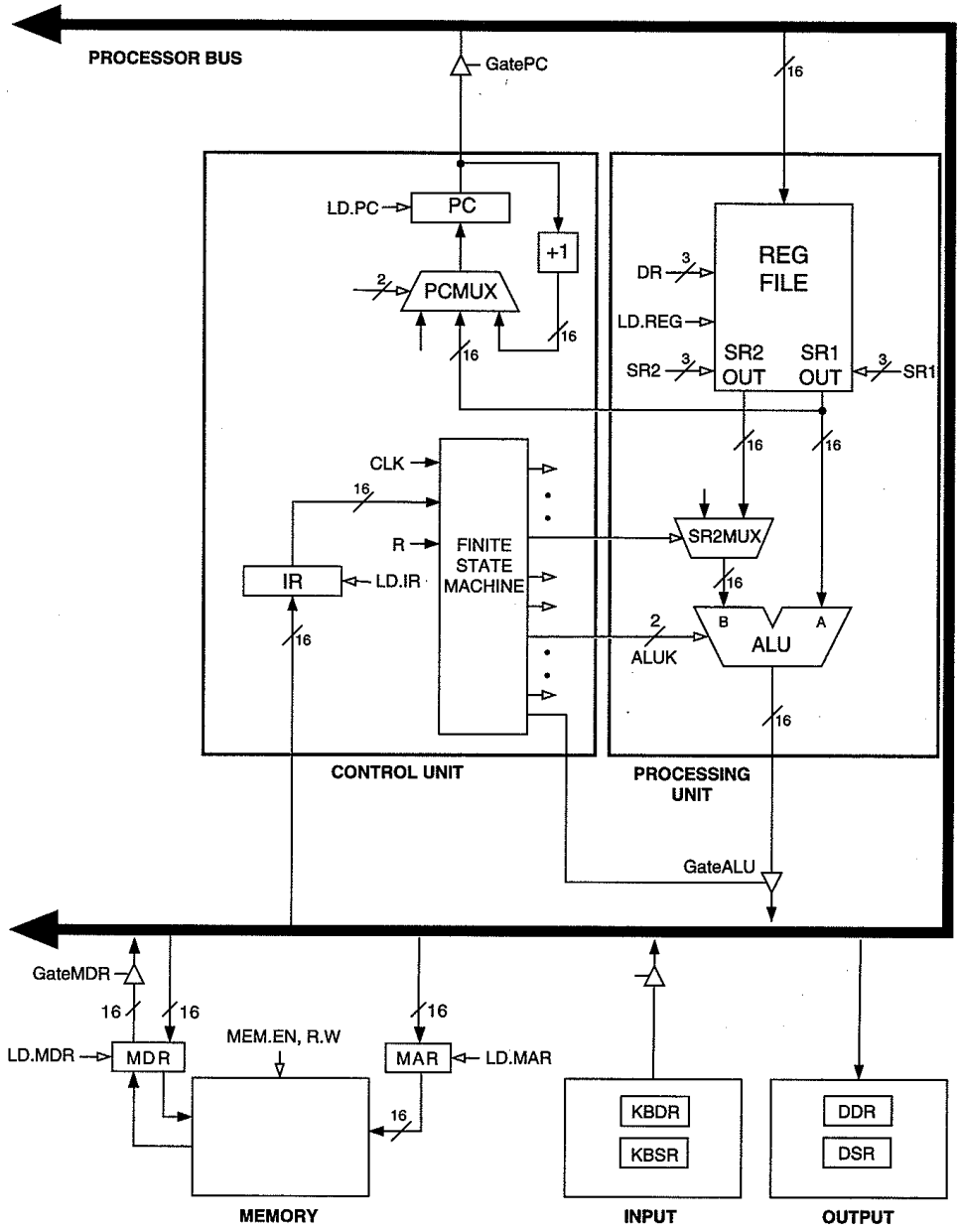
Control unit

- Directs the work of all other units
- Keeps track of which instruction is being executed, and which instruction will be processed next; for this it uses two special-purpose registers

- *Instruction register* (IR) contains current instruction being executed
- *Program counter* (PC) register keeps a pointer (address) to the next instruction to be executed
- In general, control unit can be thought of as a large FSM and the control unit cycles between different states as it directs the rest of the computer to execute a given instruction

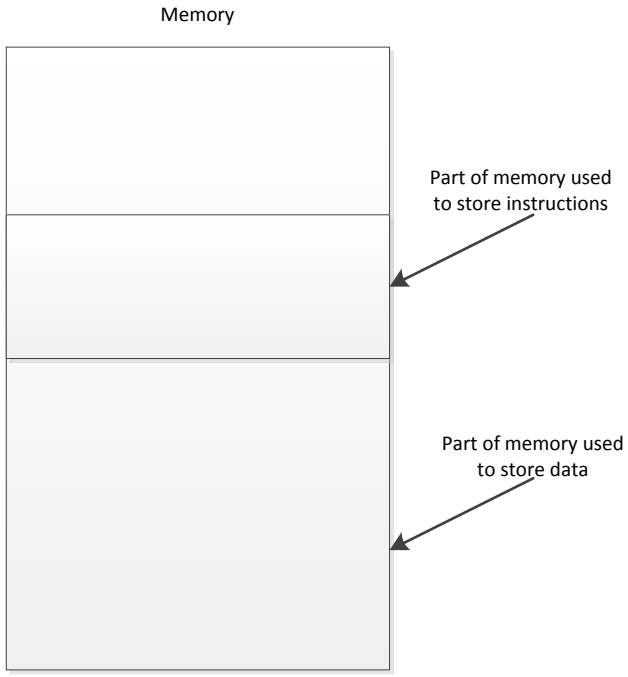
LC-3 as a von Neumann machine

- In this course we will study a computer, called Little Machine 3, or LC-3 which is an implementation of the basic von Neumann architecture:



Stored program concept

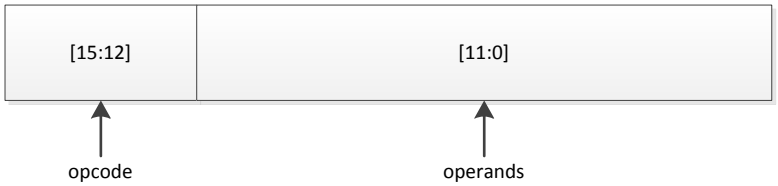
- Program is stored in some part of computer memory as a sequence of *instructions*
- Instructions are represented and stored in memory as binary words



- Control unit reads an instruction form the memory
  - Instruction address of the next instruction to be executed is stored in PC

The Instruction

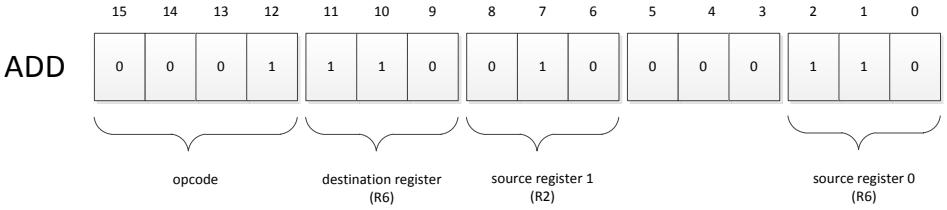
- The most basic unit of computer processing
- In the simplest form, consists of two parts
  - *Opcode* (operation code) – a portion of a machine language instruction that specifies the operation to be performed
  - *Operands* – a part of a machine language instruction that specifies the data to be operated on
- LC-3 instruction format example
  - 16-bits (one machine word)
  - The left-most 4 bits contain the opcode
  - The rest of the bits are used to encode where the operands are



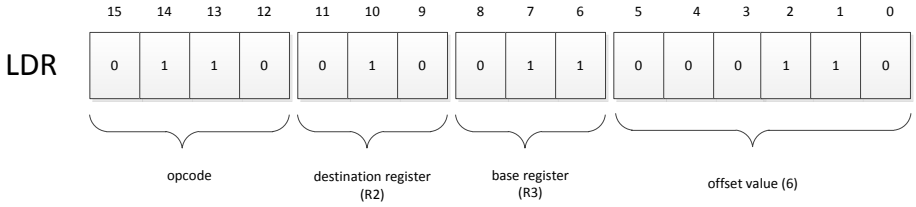
- Example: 0001110010000110
  - 0001 is an opcode
  - 110010000110 – these bits contain information about the operands
- Generally, there are 3 types of instructions:
  - Operate – perform some operation, e.g., ADD
  - Data movement, e.g., load value form memory to a register
  - Control – change the value of PC

LC-3 instruction examples (if time permits)

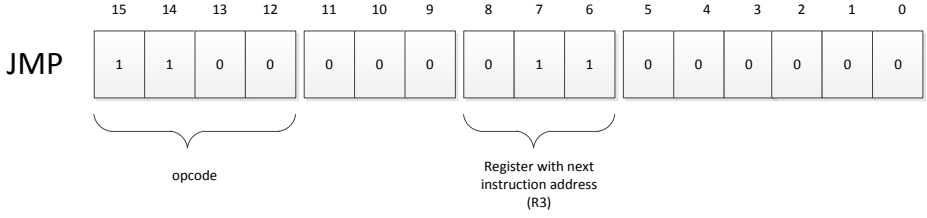
- ADD instruction
  - Requires 3 operands:
    - Two *source* operands
    - One *destination* operand
  - Action: R6←R2+R6 (values stored in registers 2 and 6 are added together and the result is placed into register 6)



- LDR instruction
  - "LD" – load, "R" – "base+offset" addressing mode
  - Action: R2←MEM[R3+6] (add value 6 (offset) to the value stored in register 3 (base) and load value from the computer memory stored at the address base+offset to the destination register 2)



- JMP instruction
  - Action: PC←R3 (load PC with a value stored in register 3)



Von Neumann instruction cycle

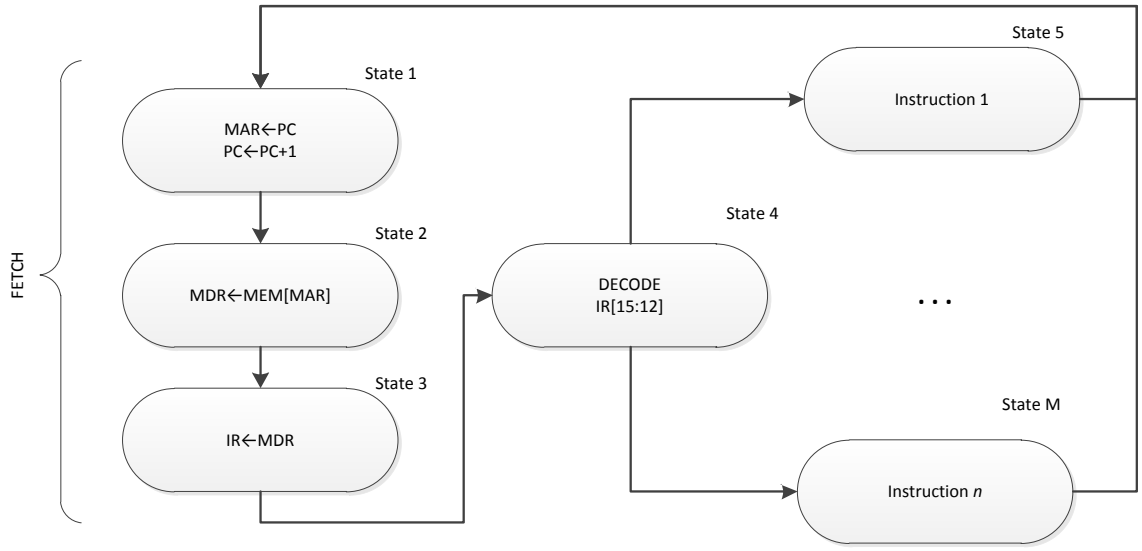
- Instructions are processed by the control unit in a systematic, step-by-step manner
- The sequence of steps in which instructions are loaded from memory and executed is called *instruction cycle*
- Each step in the sequence is referred to as a *phase*
- Fundamentally, there are 6 phases
    - FETCH (instruction)
    - DECODE
    - EVALUATE ADDRESS
    - FETCH OPERANDS
    - EXECUTE
    - STORE RESULTS
- FETCH INSTRUCTION phase
    - Obtain the next instruction from memory and store it in the IR
    - Note that the address of the next instruction to be executed is stored in the PC register
    - Proceeds in the following manner
        - MAR ← PC (memory address register is loaded with the content of PC)
        - PC ← PC + 1 (value stored in the PC is incremented by one)
        - MDR ← MEM[MAD] (interrogate memory, resulting in the instruction being placed in the MDR)
        - IR ← MDR (load the instruction from MDR to the instruction register)
    - For now, we will say that each of these steps proceeds in one *machine cycle*
    - Note that the instruction to be executed is now stored in IR and the address of the *next* instruction to be executed is stored in PC
- DECODE phase
    - The instruction stored in PC is examined in order to decide what portion of the microarchitecture needs to be involved in the execution of the instruction
    - For example, for a 4-bit opcode, this can be implemented as a 4-to-16 decoder
    - This decoder will examine bits 12-15 stored in the IR and will activate the appropriate circuitry necessary to carry out the instruction
- EVALUATE ADDRESS phase
    - Compute the address of the memory location that is needed to process the instruction
    - Some instructions do not need this phase, e.g., instructions that work directly with the registers and do not require any operands to be loaded or stored form memory
- FETCH OPERANDS phase
    - In this phase, the source operands needed to carry out the instruction are obtained from memory
    - For some instructions, this phase equals to loading values form the register file
    - For others, this phase involves loading operands from memory
- EXECUTE phase
    - Instruction is carried out

- Some instructions may not require this phase, e.g., data movement instructions for which all the work is actually done in the FETCH OPERANDS phase

- STORE RESULTS phase
  - The result is written to its designated destination
- After the 6 phases of the instruction cycle are done, the control unit begins the next instruction cycle, starting with the new FETCH (instruction) phase
  - Since the PC was previous incremented by one, it contains the pointer to the next instruction to be fetched and executed

Control of the instruction cycle

- Each step of the instruction cycle and of all its sub-steps is controlled by a FSM an abbreviated state diagram of which is shown below
  - Each state corresponds to one clock cycle of activity
  - The processing controlled by each state is shown in each node
  - Transition between states is shown by arrows
- Instruction cycle starts with the state 1 and then progresses to the next state



LC-3 FSM

- State machine consists of 52 distinct states
  - Shown on page 568 (C.2) of the textbook