Exam 1

Nathan Warner

Computer Science
Northern Illinois University
United States

Contents

Von Neumann model

- ***What is a stored-program computer? Why were they an improvement over plugboard machines?***: A stored program computer is a computer that stores programs / instructions in memory, instead of using hardware / circuits to implement programs. By doing this, changing the program meant you didn't need to change the hardware. This meant more efficient computers, less time needed to change / write programs

  A stored program computer has

  1. CPU
  2. Main memory
  3. I/O

- ***Who was John von Neumann? What is the relationship between stored-program computers and the von Neumann model? What is the relationship between the von Neumann model and the fetch-decode-execute cycle?***: J.V.N was the inventor of the stored program computer (V.N model). The V.N model refers to stored program computers, computers that employ the fetch-decode-execute cycle

- ***What is the von Neumann bottleneck? What is the relationship between the von Neumann bottleneck and the bus? How does having multiple processors help mitigate the von Neumann bottleneck?***: The single data path between the CPU and main memory is the V.N bottleneck, the bus carries data from the CPU and main memory, i.e along the V.N bottleneck. Multiple processors helps mitigate the V.N bottleneck because work / memory requests are split amongst the processors. Adds parallelism, all the data is no longer waiting at a single processor.

Core architecture

- *What is the datapath? control unit? CPU? ALU? What is the relationship between them?*: **The datapath is one of the core parts of the CPU. The datapath consists of an ALU and registers that are connected by a data bus that is also connected to main memory**

  The control unit is the second core part of the CPU, the control unit provides signals that tell the CPU components what sequenced operations it should perform.

  The CPU is the part of the computer that fetches, decodes, and executes program instructions, made up of control unit and datapath

  The Arithmetic Logic Unit (ALU) is a part of the CPU and carries out arithmetic and logical operations that are directed by the control unit.

  The CPU executes instructions using the coordination between its control unit and datapath. The control unit tells the datapath what to do; the datapath (via the ALU and registers) performs the operations and moves data; together, they fetch, decode, and execute the program's instructions.

- *What are the three types of lines on a bus? What does each of them do*:
  1. **Data lines**: Conveys bits from one device to another.
  2. **Address lines**: Determines the source or destination of the data.
  3. **Control lines**: Determines the direction of data flow, and when each device can access the bus.

- *How many address lines are required for a byte-addressable machine with 2^n bytes of memory?*: **Requires** 8 **address lines**

- *What is the difference between a point-to-point bus and a multipoint bus?*: **A point to point bus carries data from one device to another, only two devices share the bus.**:

  A multipoint bus carries data to and from many devices, many devices share the bus.

- *What is bus arbitration? Name several types of bus arbitration and the differences between them.*: **Bus arbitration refers to the protocols (algorithms) in place to resolve the conflicts that happen when multiple devices want to use the bus at the same time**
  1. **Daisy-chain**: **Permissions are passed from highest priority device to lowest priority**
  2. **Centralized parallel**: **Each device is connected to an arbitration circuit**
  3. **Distributed using self-detection**: **Devices decide who gets the bus amongst themselves**
  4. **Distributed using collision-detection**: **Any device can try to use the bus, if collisions occur, they try again.**

MARIE architecture

- *How many bits are in a MARIE data word? What data format does MARIE use internally (decimal, unsigned binary, or 2's complement)?*: **16-bits, 2's complement**
- *How many data words does MARIE have? Is MARIE byte-addressable or word-addressable? How many bits are in a MARIE address, i.e., how many bits does an address need to have a unique address for every data word?*: $\mathbf{2^{12} = 4096}$ **data words, word-addressable, 12-bits, 12-bits**
- *How many bits are in a MARIE instruction? Is MARIE a fixed word length or variable word length machine?*: **16-bits**, fixed word length
- *What do the AC, MAR, MBR, PC and IR do in MARIE? How many bits does each have? Why does each register have the number of bits that it has? (Hint: What type of data or instruction goes in each register?)*:
  1. **AC**: Accumulator, holds operands for arithmetic and results of arithmetic, 16-bits since it holds words (words are 16-bits)
  2. **MAR**: Holds an address, 12-bit because addresses are 12-bit
  3. **MBR**: Holds a word, 16-bit because words are 16-bit
  4. **PC**: Holds the address of the next instruction, 12-bit because addresses are 12-bit
  5. **IR**: Holds the current instruction, 16-bit because instructions are 16-bit
- *What is the fetch-decode-execute cycle? What happens in each step? How is the fetch-decodeexecute cycle different depending on whether the instruction contains an address operand or not?*: **The fetch-decode-execute cycle is the steps a computer goes through when running a program.**
  1. **Fetch:** Fetches instruction from memory and places into IR
  2. **Decode**: Instruction in the IR is decoded to determine what needs to be done next
  3. **Execute**: The instruction is executed

  If the instruction requires an operand, an additional step is done between steps 2 and 3 (fetch-operand). The operand is fetched from memory and placed inside the MBR.
- *Which MARIE instructions need a fetch operand step? What is the maximum number of fetch operand steps a MARIE instruction can have? Why?*: **At most one fetch-operand step, since all instructions either have zero or one operands**.
- *What is RTL? What is a microoperation? Which steps in the execution cycle (fetch, decode, fetch operand, execute) use the same RTL for every instruction? What does the RTL for fetch do? the RTL for fetch operand? the RTL for execute? Why does the decode step not need any RTL?*: **Each instruction is made up of a sequence of smaller instructions called microoperations. The microoperations are specified using the register-transfer language (RTL). Fetch, decode, fetch operand use the same RTL for every instruction**
- *How many bus addresses does MARIE have?*: **7**
- *What are the parts of a MARIE instruction? How many bits does each part have? Why?*: The opcode (4 bits), MARIE has 12 instructions, need 4 bits = $2^4 = 16$ possible opcodes. and the address operand (12-bits)

- *Make sure you understand why the RTL for each instruction works. For example, why does the RTL for JUMP load the operand into the PC? Why does SKIPCOND not contain an address to skip to?*: **Only 12-bits for the operand. Since addreses are 12-bits, if we supplied an address to jump to , there would be no room for the condition. We use bits 10 and 11 for the condition**

MARIE execution cycle

- **Because some lines of RTL for the load instruction (the first two) are already done during the fetch-decode-fetch operand steps, they are redundant**

Indirect addressing

- *What is indirect addressing? Which MARIE instruction(s) use indirect addressing?*: **In indirect addressing, the address of the address of the operand is given in the instruction, ADDI and JUMPI use it**

- *How do you call a subroutine in MARIE? How do you return from a subroutine? Why can't MARIE handle recursive calls (instances where a function calls itself)?*: **Use the JNS (jump and store) instruction. To return, jump to the address stored in the address given to JNS in the subroutine call. Can't handle recursion because there is only room for one return address**, for recursion each call would need to store its return address.

- *What is the difference between the DEC and HEX assembler directives? (Assembler directives are commands that do something inside the assembler but not in the executable program, so they do not create RTL. In this case the assembler directives assign memory.)*:

  The `DEC` and `HEX` directives both tell the assembler to reserve a memory location and store a constant value there. The difference is in the number system used to interpret that value. The `DEC` directive stores the value as a decimal (base 10) number, while the `HEX` directive stores the value as a hexadecimal (base 16) number. These directives do not generate executable instructions; they only define initial data values in memory.

Hardwired implementation

- *What is hardwired control? What is microprogramming? What is the difference between them?*: **In hardwired, circuits (digital logic components) are used to generate the control signals. In Microprogramming , execution of microcode instructions produces control signal changes.**

- *What is a signal line? What is a datapath address? How many signal lines (input and output) does a machine need if there are 2^n addresses on the datapath? What is a timing signal? How many timing signals are needed to handle all the instructions, including the indirect addressing instructions and the other advanced ones in section 4e (i.e., all the instructions on the MARIE summary sheet? Why do some instructions need more timing signals than others? What does the counter reset signal do? When is it triggered? What would happen if we didn't have a counter reset signal? Do these terms refer to a hardwired machine or a microprogrammed machine?*: **A signal line is a wire used to carry a control signal (bit). A datapath address is the bus address of a register or main memory, each register and main memory has a unique address along the datapath. If there are 2^n address along the datapath, we need n signal lines for the input, and n for the output. A timing signal is a control signal used to indicate the clock tick number, used to determine which line of RTL we need to execute. A MARIE instruction has at most 7 lines of RTL, so we have 7 timing signals. Some instructions require more timing signals because they require more lines of RTL. An instruction only needs enough timing signals for each line of RTL (one per line). The counter reset signal resets the binary counter that produces the timing signals (back to zero), it is triggered after the execute step of the instruction. If we did not have counter reset, the counter would not be reset for the next instruction, and incorrect timing signals would be produced. These concepts refer to a hardwired machine**

- *What do the following symbols represent: P0-P5, T0-T7, A0-A3, Cr?*: address signals, timing signals, ALU signals, counter reset

Microcode implementation

- *What is firmware? How often are microcode instructions retrieved from the firmware? Why do MARIE microinstructions have space for two microoperations? What goes in the second one if it's not used in a given microinstruction? What are the Jump and Dest fields and how are they used?*: **Firmware is the software stored in ROM that controls the hardware, one microcode instruction is retrieved during each clock cycle**

  Marie microinstructions have space for two microoperations, the first one is the opcode for the microinstruction, and the second is used to hold the opcode of the MARIE instruction, in some microinstructions we compare the microopcode to the MARIE opcode, so that we know which microoperation to go to to start executing the MARIE instruction. If the microinstruction is not a comparision, we put NOP as the second microoperation. The jump bit is either one or zero, if its set to one, we jump to the address in the destination field. The dest field is the address that we jump to if the jump bit is on. After the first four microinstructions (fetch and decode) are executed, we go through the jump table, and find the microinstruction that has the same opcode as the marie instruction. That instruction will have the jump bit set on, and the dest field has the address of the start of the execute step for that instruction.

- *Which steps of the fetch-decode-execute cycle are located before the jump table in MARIE's microcode? Which lines of the microcode implement fetch? decode? fetch operand? execute? What does the jump table contain? What is at the addresses jumped to by the jump table? Why does the last line of each of the jumped-to sections have the jump flag set? Where does each of these lines jump to?*: **Fetch and decode are before the jump table, lines 1-3 implement fetch, line 4 implements decode. The lines of microcode that implement fetch-operand and execute are found at the jump address. The jump table contains microinstructions that compare the rightmost 4 bits of the IR (MARIE instruction opcode) to microopcodes, it also contains the jump dest address, which is the address of the start of the execute step for the instruction found in microOp2. The last line has the jump flag set, and jumps to address zero, which is the beginning of the microprogram (fetch-decode-...)**.

Important information

- **V.N bottleneck**: Single datapath between CPU and memorn may
- **CPU**: Fetches, decodes, and executes program instructions
- **Two principal parts of thheh CPU**
  1. **Datapath**: Consists of ALU and storage units (registers), connected by a data bus that is also connected to memory.
  2. **Control unit**: Produces control signals that tell the CPU components which operations to perform
- **Bus**: Set of wires (lines) that convey signal bits across each line.
  - **Point to point bus**: Only two devices are connected to the bus
  - **Multipoint bus**: Many devices are connected to the bus
- **MARIE bus**: MARIE uses a single shared 16-bit bus for both data and addresses (time-multiplexed).

  That means:
  - The same 16 wires carry either addresses or data at different times.
  - Control signals decide what's currently being sent.
- **Send and receive data through the bus**: When one component needs to send or receive data:
  - The Control Unit activates that device's bus address line.
  - That component either places data on the bus (output) or reads from it (input).
- **ISA**: A computer's instruction set architecture (ISA) specifies the format of its instructions and the primitive operations that the machine can perform. The ISA is an interface between a computer's hardware and its software. Some ISAs include hundreds of different instructions for processing data and controlling program execution.

  The MARIE ISA has only thirteen instructions
- **Control unit**: Ensures that each instruction is executed in sequence, making sure that data flows to the correct components as each instruction is executed.
  - **Hardwired control unit**: Digit logic components are used to select the to and from components and ensure that data flows to them at the correct time
  - **Microprogrammed control**: Machine contains a small program in ROM in a component called the microcontroller. The microprogram contains RTL for each instruction. To execute an instruction, the microcontroller interprets the corresponding piece of the microprogram

    The microcontroller (microprogram control unit) is the hardware that fetches, decodes, and executes microinstructions... just like the CPU.

Microprogrammed machines have the same control lines, the patterns for the control lines are stored as data in the control store.

- **Microprogramming**: Execution of microcode instructions produces control signal changes. The microprogram converts each microcode instruction into control signals.

  Microprogram stored in firmware. The piece of ROM that holds the microprogram is called the control store. One microinstruction retrieved during each clock cycle.

  Doesn't remove the control unit, changes its implementation. Instead of AND/OR gates and flip-flops generating control signals, you have

  – The microinstructions in the control store.
  – Microsequencer (microcontroller / microprogram control unit) that fetches, decodes, and executes the microinstructions

  Note: A microprogrammed control unit still requires circuits.

Aspect	**Hardwired Control**	**Microprogrammed Control**
Control logic	Implemented entirely by fixed logic circuits	Implemented mostly as data stored in the control store (ROM or PLA)
Still needs circuits?	Yes (for all control signals and timing)	Yes (for the control store, microinstruction register, sequencing logic, etc.)

  So the microprogrammed control unit is still a circuit, but one that reads control words from memory instead of generating them entirely through logic.

- **Generic microsequencer**:

  1. IR $\rightarrow$ microinstruction address generator $\rightarrow$ control store $\rightarrow$ microinstruction buffer $\rightarrow$ microinstruction decoder $\rightarrow$ control signals