Lecture 25: Microprogramming

- The Microprogram
- Encoding instructions
- Decoding Instructions

The Microprogram

- microprogram from handout

Fetch and Decode

0: MAR := PC; READ; // gets low-order 13 bits
1: READ; // data returned in MBR
2: PC := PC + 1;
3: IR := MBR; if N then goto 25 // check bit 0 of opcode
4: TMP := shift(IR + IR); if N then goto 16 // check bit 1
5: TMP := TMP; if N then goto 10; // check bit 2

- Bits are numbered left-to-right! (normally the sign bit is bit 15, here it is bit 0).
- Reading takes two microinstruction cycles.
- Adding IR to IR sets the high bit, setting N
- The shift sets up the value in TMP to be checked for bit 2 being zero.
- Additional decoding (bit checking) is done at locations 16 and 25.

ADD

6: MAR := IR; READ; // ADD (000)
7: READ;
8: ACC := MBR + ACC;
9: goto 0;

- IR is copied to MAR – this grabs the lower 13 bits (the ones that contain the address of our operand).
- The result of the read is returned in MBR
- The operand is then added to the accumulator.
- At the end, the goto jumps back to the fetch portion of the microcode to get the next instruction.
SUB

10: MAR := IR; READ;  // SUB (001)
11: READ;
12: ACC := ACC + 1;
13: TMP := com(MBR);   // 1's complement
14: ACC := ACC + TMP;
15: goto 0;

- The read is the same as for ADD
- The accumulator is incremented by one
  - remember, 1 is actually a register always set to that value!
- The value in MBR (the item we read) is complemented.
- The complemented value is added to ACC
- Why did we add one to ACC to start with? Because com only takes the 1’s complement. To add a 2’s complement number, we need to add the extra one!

LOAD

16: TMP := TMP; if N then goto 21 // check bit 2
17: MAR := IR; READ;  // LOAD (010)
18: READ;
19: ACC := MBR;
20: goto 0;

- Before we can execute the microcode for LOAD, we need to check if bit 2 is clear.
- We read the data as before.
- We then copy the data into ACC from MBR.

STORE

21: MAR := IR;  // STORE (011)
22: MBR := ACC; WRITE;
23: WRITE;
24: goto 0;

- For STORE, IR holds the address we are writing to.
- The address is copied into MAR
- ACC is copied into MBR (that’s the data we are writing).
- WRITE takes two microinstruction cycles (just like read).

More Decoding

25: TMP := lshift(IR+IR); if N then goto 0;  // No opcodes 11...
26: TMP := TMP; if N then goto 29

- We need to decode the last bit to see what type of jump instruction it is.
- We check the bit using lshift(IR+IR) like we did at the start of the microprogram.
**JUMP**

27: \( PC := \text{and}(IR, \text{AMASK}); \)  
28: goto 0;

- JUMP is an unconditional jump
- Our special AMASK register is used to mask out the Opcode before copying the address into the PC
- Why didn’t we have to do this for MAR?
  - MAR is a 13 bit register so the upper 3 bits are not stored
- What value is in AMASK?

**JZER**

29: \( ACC := ACC; \) if Z then goto 27:  
30: goto 0;

- JZER copies ACC to ACC so that the flags will be set depending on the value in ACC.
- If zero, it jumps back to the unconditional jump code:
  27: \( PC := \text{and}(IR, \text{AMASK}); \)  
  28: goto 0;

**MicroInstruction Format**

- review format (handout)

- Instruction encoding example:
  2: \( PC := PC + 1 \)
- Fields:
  - MUX: 0 – left input is A-latch (MBR does not appear in this instruction)
  - COND: 00 – no jump
  - ALU: 00 – addition
  - SH: 00 – don’t shift
  - MBR: 0 – we’re not storing anything in MBR
  - MAR: 0 – we’re not storing anything in MAR
  - RD: 0 – not reading
  - WR: 0 – not writing
  - ST: 1 – we are storing something in a register (if there is := in the instruction and the destination is not MAR or MBR then ST should be set)
  - C: 001 (PC is register 1)
  - B-latch: 101 (1 is register 5)
  - A-latch: 001
  - ADDR: doesn’t matter (no read, no write)
- Answer:
  0 00 00 00 0 0 0 0 1 001 101 001 000000
  - spacing is for readability!
• Another example:
  13: TMP := c0m(MBR)
• Fields:
  MUX: 1 – we’re getting our input from MBR
  COND: 00 – no jump
  ALU: 11 – complement
  SH: 00 – don’t shift
  MBR: 0 – we’re not storing anything in MBR
  MAR: 0 – we’re not storing anything in MAR
  RD: 0 – not reading
  WR: 0 – not writing
  ST: 1 – we are storing something in a general register.
  C: 011 (TMP is register 3)
  B-latch: doesn’t matter (not using B-latch)
  A-latch: doesn’t matter (not using A-latch)
  ADDR: doesn’t matter (no read, no write)
• Answer:
  1 00 11 00 0 0 0 1 011 000 000 000000

• More than one MAL instruction per microinstruction!:
  16: TMP := TMP; if N then goto 21
• Fields:
  MUX: 0 – we’re getting our input from A-latch
  COND: 01 – jump if N=1
  ALU: 10 – passing A-latch value through
  SH: 00 – don’t shift
  MBR: 0 – we’re not storing anything in MBR
  MAR: 0 – we’re not storing anything in MAR
  RD: 0 – not reading
  WR: 0 – not writing
  ST: 1 – we are storing something in a general register
  C: 011 (TMP is register 3)
  B-latch: doesn’t matter (not using B-latch)
  A-latch: 011 (our operand is TMP)
  ADDR: 010101 – jumping to address 21
• Answer:
  0 01 10 00 0 0 0 1 011 000 011 010101

• Another example:
  17: MAR := IR; READ;
• Fields:
  MUX: 1 – we’re getting our input from MBR
  COND: 00 – no jump
  ALU: 10 – passing A-latch value through
  SH: 00 – don’t shift
  MBR: 0 – we’re not storing anything in MBR
  MAR: 0 – we’re not storing in MAR
  RD: 1 – we are reading
  WR: 0 – not writing
  ST: 0 – we are NOT storing anything in a general register
  C: doesn’t matter (we’re storing in MAR)
  B-latch: doesn’t matter (not using B-latch)
  A-latch: 010 (our operand is IR)
  ADDR: doesn’t matter
• Answer:
  1 00 10 00 0 1 10 000 000 010 000000

• Another example:
  27: PC := and(IR, AMASK)
• Fields:
  MUX: 0 – we’re getting our input from A-latch
  COND: 00 – no jump
  ALU: 01 – AND
  SH: 00 – don’t shift
  MBR: 0 – we’re not storing anything in MBR
  MAR: 0 – we’re not storing into MAR
  RD: 0 – not reading
  WR: 0 – not writing
  ST: 1 – we are storing into a general register
  C: 001 (PC = 1)
  B-latch: 100 (our second operand is AMASK)
  A-latch: 010 (our first operand is IR)
  ADDR: doesn’t matter
• Answer:
  0 00 01 00 0 0 0 1 001 100 010 000000
Decoding

• Decoding hints:
  – break the binary up into the fields as shown
  – an unused field is not necessarily going to be zero! Any non applicable fields (addresses if not jumping, B-latch values for single operand instructions) should be ignored.

• Decode Example

 0 0 1 0 0 0 1 0 0 0 0 1 0 1 0 0 1 0 1 0 0 0 0 0 0 0 0 0

Fields:
  MUX: 0 – we’re getting our input from A-latch
  COND: 01 – jump if N
  ALU: 00 – ADD
  SH: 01 – shift left
  MBR: 0 – we’re not storing anything in MBR
  MAR: 0 – we’re not storing into MAR
  RD: 0 – not reading
  WR: 0 – not writing
  ST: 1 – we are storing into a general register
  C: 011 – (3 – TMP register)
  B-latch: 010 (our second operand is IR)
  A-latch: 010 (our first operand is IR)
  ADDR: 000000

• Answer:
  TMP := lshift(IR + IR); if N then goto 0

• Decode Example

 0 1 1 1 0 0 0 0 0 0 0 0 1 1 1 0 0 1 1 0 0 1 1 0 0

Fields:
  MUX: 0 – we’re getting our input from A-latch
  COND: 11 – unconditional jump
  ALU: 10 – pass through
  SH: 00 – not shifting
  MBR: 0 – we’re not storing anything in MBR
  MAR: 0 – we’re not storing into MAR
  RD: 0 – not reading
  WR: 0 – not writing
  ST: 0 – not storing into a general register
  C: 011 – (3 – TMP register)
  B-latch: 101 (1 register)
  A-latch: 010 (our second operand is IR)
  ADDR: 000000

• Answer:
  go to 0

In-Class Exercise

• Create the binary microinstruction for the following:
  MBR := ACC + 1; WRITE;

• Write MAL code for the following:

 0 1 0 1 1 0 1 0 0 0 0 1 0 1 1 0 1 0 1 0 0 1 1 0 0 1 0 1 1 0 0 1 1 0 0

• Answer: ???
MBR := ACC + 1; WRITE;

- **Fields:**
  - MUX: 0 – we’re getting our input from A-latch
  - COND: 00 – no jump
  - ALU: 00 – we’re adding
  - SH: 00 – don’t shift
  - MBR: 1 – we’re storing our result in MBR
  - MAR: 0 – we’re not storing anything in MAR
  - RD: 0 – not reading
  - WR: 1 – we’re writing!
  - ST: 0 – we’re not storing into a general register.
  - B: doesn’t matter (storing into MBR)
  - ADDR: doesn’t matter.

- **Answer:**
  0 0 0 0 0 0 1 0 0 1 0 0 0 0 1 0 1 0 0 0 0 0 0

---

0 1 0 1 0 1 0 0 0 1 0 1 1 0 0 1 0 0 1 1 0 0 1 0 0

- **Fields:**
  - MUX: 0 – we’re getting our input from A-latch
  - COND: 10 – jump if Z=1
  - ALU: 10 – pass through
  - SH: 10 – right shift
  - MBR: 0 – we’re not storing anything in MBR
  - MAR: 0 – we’re not storing into MAR
  - RD: 0 – not reading
  - WR: 0 – not writing
  - ST: 1 – storing into a general register
  - C: 011 – (3 – TMP register)
  - B: 001 (PC)
  - ADDR: 001100

- **Answer:**
  0 1 0 1 0 0 0 1 0 0 1 0 1 0 0 0 1 1 0 0 1 0 0 1 0 0 1 1 0 0 1 0 0

  TMP := rshift(TMP); If Z then goto 12