CS 366 - Computer Architecture II

The Mythsim Data Path

The data path can be divided into 3 parts:

1. Register Set
2. ALU - Arithmetic Logic Unit
3. Memory Interface

The data flows from the Register Set to the ALU to the Memory Interface and back to the Register Set. There may be two values flowing from the Register Set to the ALU, but only one value between any two other parts. Note the Memory Interface will interact with the Memory Unit, but that will be discussed later.

The register set contains 8 registers, named r0, r1, ... and r7. r0, r1, r2 and r3 are the general purpose registers used by the programs executing on the machine. r4, r5 and r6 are temporary registers used by the Mythsim machine itself. r7 is used as the program counter for the programs executing on the machine. The register set has 10 control inputs (or control lines), named Asel, Bsel, r0write, r1write, ... and r7write. Between the Register Set and the ALU there are two data busses, the A bus and the B bus. The Asel control line determines which register value is sent along the A bus. The Bsel control line determines which register value is sent along the B bus. Between the Memory Interface and the Register Set there is one data bus. The r0write - r7write control lines determine which register(s) will store the value on the bus. This value can be stored in multiple registers or in no register.

The ALU performs the following operations

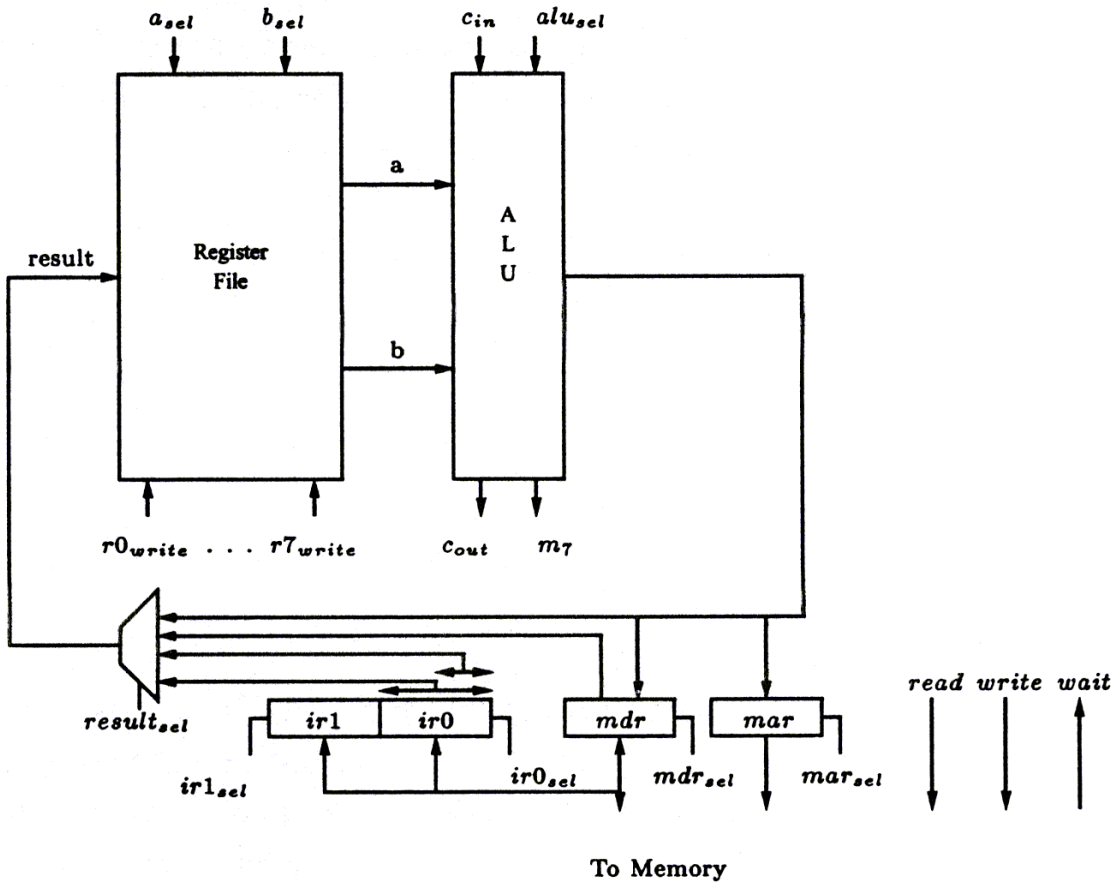
- NOT: !a
- OR: a | b
- AND: a & b
- XOR: a ⊕ b
- ADD: a + b + cin
- SUB: a + !b + cin
- ADDA: a + cin
- SUBA: a - 1 + cin

The ALU has two control inputs (or control lines), named cin and aluSel. The aluSel control line determines the operation to the performed by the ALU. The cin control line is used with two complement addition and can have a value of either 0 or 1. When cin is 1 and aluSel is set to SUB, the operation preformed is a - b. The ALU has three status outputs (or status lines), named cout, m7 and v. The cout status line informs of any carry out as a result of the operation. The m7 status line informs of the sign of the result of the operation. The v status line informs if overflow occurred during the operation.

The Memory Interface contains three registers, named mar (Memory Address Register), mdr (Memory Data Register) and ir (Instruction Register). When a read or write with the memory is to occur, the address in memory where the read or write is to occur is stored in the mar. When a data read or write is to occur, the data value is put into the mdr before the write and retrieved from the mdr after a read. When an instruction read is to occur,

the instruction value is retrieved from the ir after the instruction read. The Memory Interface has 5 control inputs (or control line), named marSel, mdrSel, ir0sel, ir1sel and resultSel. The marSel control line determines when a new address is loaded into the mar from the output of the ALU. The mdrSel control line determine when a new value is loaded into the mdr from either the output of the ALU or from memory. The ir0sel control line determines when a new instruction is loaded into the upper 8 bits of the ir. While the ir1sel control line determines when a new instruction is loaded into the lower 8 bits of the ir. The resultSel control line determines which value is sent to the Register Set. The possible values are the output from the ALU, the value in the mdr, a constant value contained in the current instruction in the ir.

The Mythsim machine has two other control lines and one other status line. These interact with the Memory Unit of the machine and control information passing between the Memory Interface and the Memory Unit. The two control lines are named read and write. The status line is named wait. The read and write control lines inform the Memory Unit that the Memory Interface is ready for either a read or write operation. The wait status line informs the machine that the Memory unit is still performing a read or write operation and the not to proceed until it is finished.



CS 366 - Computer Architecture II

Summary of Control Lines:

- Asel
- Bsel
- r0write
- r1write
- r2write
- r3write
- r4write
- r4write
- r6write
- r7write
- cin
- aluSel
- marSel
- mdrSel
- ir0sel
- ir1sel
- resultSel
- read
- write

Summary of Status Lines:

- cout
- m7
- v
- wait

CS 366 - Computer Architecture II

Fetch/Execute Cycle

Let us assume that a C/C++ program contains the following instruction:

```
x = y + 7;
```

Assuming the value for x is stored at memory position 120 and the value for y is stored at memory position 110, the corresponding assembly language code might be as follows:

```
Set     r0, 110       // set register 0 to the address for y
Load    r1, r0        // read from memory the value whose address is stored in
                      // register 0 and put this value in register 1

Set     r2, 7         // set register 2 to the value 7

Add     r3, r1, r2    // register 3 is set to the result of register 1 plus register 2

Set     r0, 120       // set register 0 to the address for x
Store   r3, r0        // store the value in register 3 at the memory address stored in
                      // register 0
```

When a program is being executed, the instructions are loaded into memory and the program counter (register 7) is set to the address in memory where the first instruction is located. In Mythsim, each instruction is 16 bits long. Assume the above code is loaded into memory starting at position 20. The first instruction would be stored in addresses 20 and 21, the second in addresses 22 and 23, the third in addresses 24 and 25, etc. We would first load the program counter (register 7) with 20 (the address of the first instruction).

To execute a program the Mythsim machine runs the Fetch/Execute cycle. The machine will first fetch the next instruction from memory (as determined by the program counter) and store this instruction in the ir. Then the machine would execute the operation(s) needed for that instruction. Let us first look at how the fetch is performed.

In Mythsim, each instruction is 16 bits long and we can only read or write 8 bits at once, so each fetch would require 2 memory reads. To fetch the first (or lower) 8 bits of the instruction, we would need to pass the address in the program counter (register 7) through the ALU (without modifying it) and store it in the mar. Then the machine would need to perform a read operation and store this value into the lower 8 bit of the ir. It would then need to increment the program counter to be ready to fetch the upper 8 bits of the instruction. Finally is would repeat those steps to retrieve to upper 8 bits from memory.

The control lines for each fetch would be set as follows:

1. Pass r7 though the ALU and store it in the mar. Note: ORing a value with itself will pass it through the ALU without modifying it.
   asel = 7; bsel = 7; aluSel = 1(OR); marSel = 1(LOAD)
2. Read the value from memory and store in the lower 8 bits of the ir
   read = 1; ir0sel = 1 (LOAD from Memory)
3. Repeat/Goto step 2 while the wait status line is on.
4. Increment the program counter
   asel = 7, cin = 1; aluSel = 6 (ADDA); resultSel = 0 (Output from ALU); r7write = 1
5. Pass r7 though the ALU and store it in the mar.
   asel = 7; bsel = 7; aluSel = 1(OR); marSel = 1(LOAD)
6. Read the value from memory and store in the upper 8 bits of the ir
   read = 1; ir1sel = 1 (LOAD from Memory)
7. Repeat/Goto step 6 while the wait status line is on.
8. Increment the program counter
   asel = 7, cin = 1; aluSel = 6 (ADDA); resultSel = 0 (Output from ALU); r7write = 1

After each instruction is fetched, the control unit will decode instruction and set the control lines as needed for the instruction. This is the execute portion of the fetch/execute cycle. The control lines as each instruction is executed are shown below.

```
Set   r0, 110        // set register 0 to the address for y
```

1. resultSel = 3 (8 bit literal value from ir); r0write = 1

```
Load  r1, r0         // read from memory the value whose address is stored in
                     // register 0 and put this value in register 1
```

1. Pass r0 through the ALU and store in the mar.
   asel = 0; bsel = 0; aluSel = 1(OR); marSel = 1(LOAD)
2. Read the value from memory and store in the mdr
   read = 1; mdrSel = 2 (LOAD from Memory)
3. Repeat/Goto step 2 while the wait status line is on.
4. Put the value in the mdr into r1
   resultSel = 1 (value from mdr); r1write = 1

```
Set   r2, 7          // set register 2 to the value 7
```

1. resultSel = 3 (8 bit literal value from ir); r2write = 1

CS 366 - Computer Architecture II

Add r3, r1, r2 // register 3 is set to the result of register 1 plus register 2

1. asel = 1; bsel = 2; aluSel = 4 (ADD); resultSel = 0 (Output from ALU); r3write = 1

Set r0, 120 // set register 0 to the address for x

1. resultSel = 3 (8 bit literal value from ir); r0write = 1

Store r3, r0 // store the value in register 3 at the memory address stored in // register 0

1. Pass r0 through the ALU and store in the mar.
   asel = 0; bsel = 0; aluSel = 1(OR); marSel = 1(LOAD)
2. Pass r3 through the ALU and store in the mdr
   asel = 3; bsel = 3; aluSel = 1(OR); mdrSel = 1(LOAD from ALU)
3. Write the value to memory
   write = 1
4. Repeat/Goto step 3 while the wait status line is on.

CS 366 - Computer Architecture II

Additional Control and Status Lines

So far we have, we have had the values on the asel and bsel control lines determine what values are taken from the registers. However, in the instruction **Add r3, r1,r2**, how does the asel control line know the have the value of 1? The same goes the bsel and the r3write control lines. How does the information get from the ir to the control lines? First, we will need to understand to form that instructions are encoded in the instruction register. Each instruction is 16 bits long (retrieved from memory in two 8 bit chunks). The bits are numbered from 15 to 0 from left to right as follows:

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

Most significant Least significant

Bits 15 through 10 are used to hold the operation code (called the "opcode" for short). The opcode tells the control unit which operation to perform. Since there are 6 bits in the opcode, there can be 64 different operations. See below as to how this value is retrieved from the instruction register.

Bits 9 and 8 are used to hold any Destination Register information. If the instruction specifies that some information will be stored in a register, that register number will be specified here. This register is normally called the register "i" or "ri". See below as to how this value is retrieved from the instruction register.

Bits 7 and 6 are used to hold any Source Register A information. If the instruction specified that the value from some register is to be send only the A bus from the register set to the ALU, that register number will be specified here. This register is normally called register "j" or "rj". See below as to how this value is retrieved from the instruction register.

Bits 5 and 4 are used to hold any Source Register B information. If the instruction specified that the value from some register is to be send only the B bus from the register set to the ALU, that register number will be specified here. This register is normally called register "k" or "rk". See below as to how this value is retrieved from the instruction register.

Bits 3 through 0 are used to hold a 4 bit constant value. This 4 bit value is sign extended to an 8 bit value. This constant can hold values from -8 to 7. This value is retrieved from the instruction register by setting the resultSel control line to CONST4.

The above fields cause the instruction to appear as follows:

opcode bits 15 to 10	ri bits 9 to 8	rj bits 7 to 6	rk bits 5 to 4	4 bit constant bits 3 to 0

The instruction could have an alternative format. This format only has three fields instead of five. Bits 7 to 0 are used to hold an 8 bit constant value in the range from -128

to 127. This value is retrieved from the instruction register by setting the resultSel control line to CONST8. The three field instruction format appears as follows:

opcode bits 15 to 10	ri bits 9 to 8	8 bit constant bits 3 to 0

The opcode, ri, rj and rk fields are accessed via new status lines called iropcode, irri, irrj and irrk. The iropcode status line is brought directly into the control unit to add with decoding of the instruction. The register status lines are used with three additional control lines called risel, rjsel and rksel.

risel is used when the ir contains information about which register should be written to. The risel control line is combined with bits from the ir (which give the register to write to) and the r0write, r1write, r2write and r3write control lines to produce the vr0write, vr1write, vr2write and vr3write control lines. If risel is on, one of the vrXwrite control lines is on based on a value in the instruction register. If risel is off, the vrXwrite control lines contain the values of the appropriate rXwrite control lines. The vrXwrite control lines would replace the appropriate rXwrite control lines connected to the register set. The information from the instruction register would be considered a new status line, called irri.

rjsel and rksel work in a similar manner to risel. The rjsel control line combines the asel control line with bits from the ir to produce the vasel control line. If rjsel is on, the vasel control line value is determined by a value in the instruction register. The vasel control line would replace the asel control line connected to the register set. The information from the instruction register would be considered a new status line, called irrj. The rksel control line combines the bsel control line with bits from the ir to produce the vbsel control line. If rksel is on, the vbsel control line value is determined by a value in the instruction register. The vbsel control line would replace the bsel control line connected to the register set. The information from the instruction register would be considered a new status line, called irrk.

The instruction Add r3, r1, r2 would have the following control lines set

- rjsel = 1; rksel = 1; aluSel = 4 (ADD); resultSel = 0 (Output from ALU); risel = 1

These would be the same control lines for any add operation no matter which registers are being used.

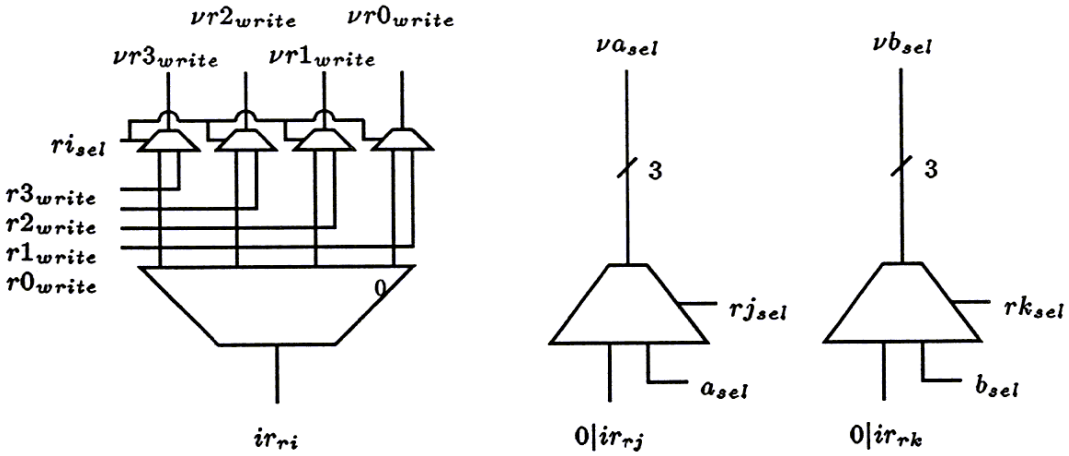
New Control Lines

- risel
- rjsel
- rksel
- vr0write
- vr1write
- vr2write
- vr3write
- vasel
- vbsel

New Status lines

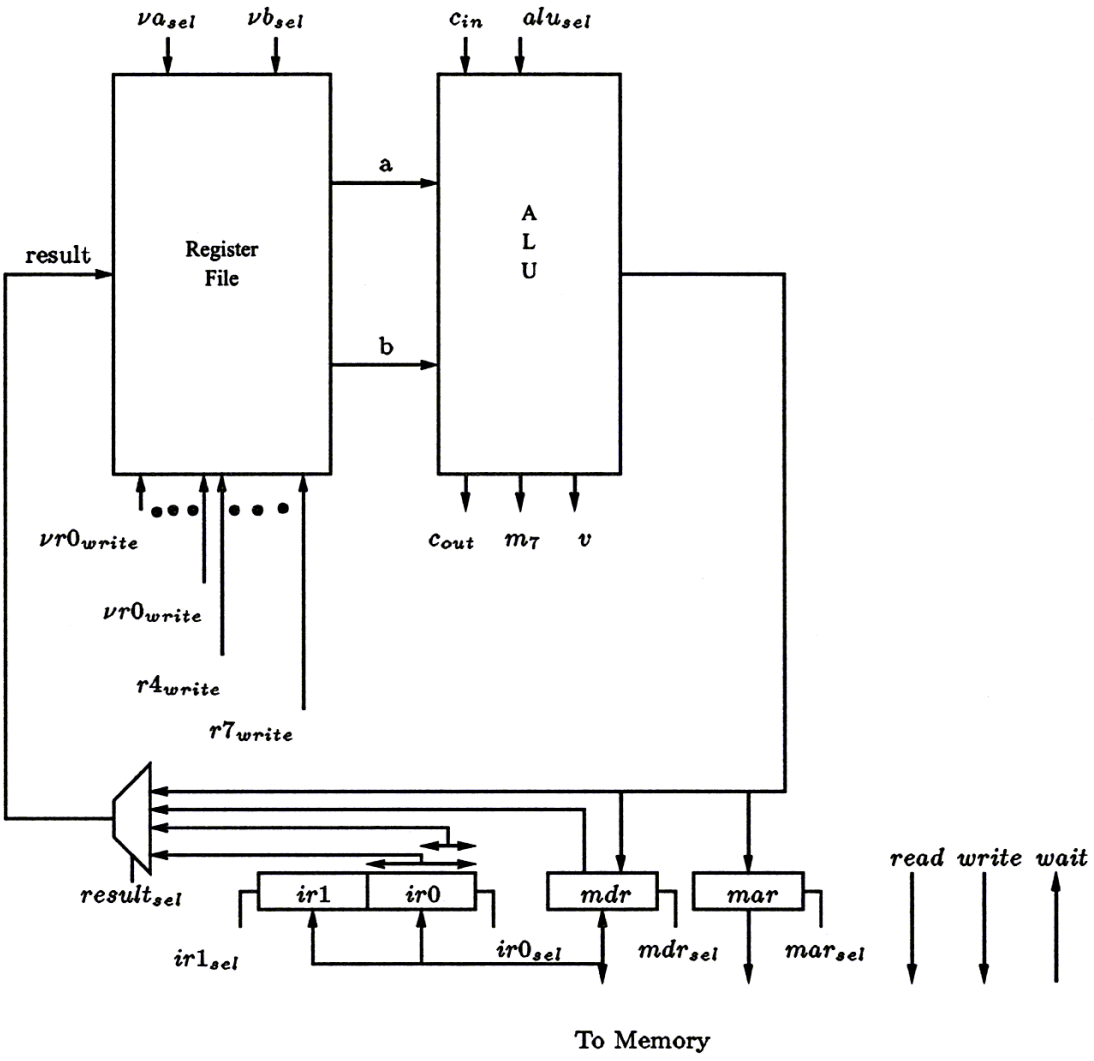
- irri
- irrj
- irrk

The circuitry that uses these new control lines and status lines is as follows:



Adding these changes the entire datapath will be:

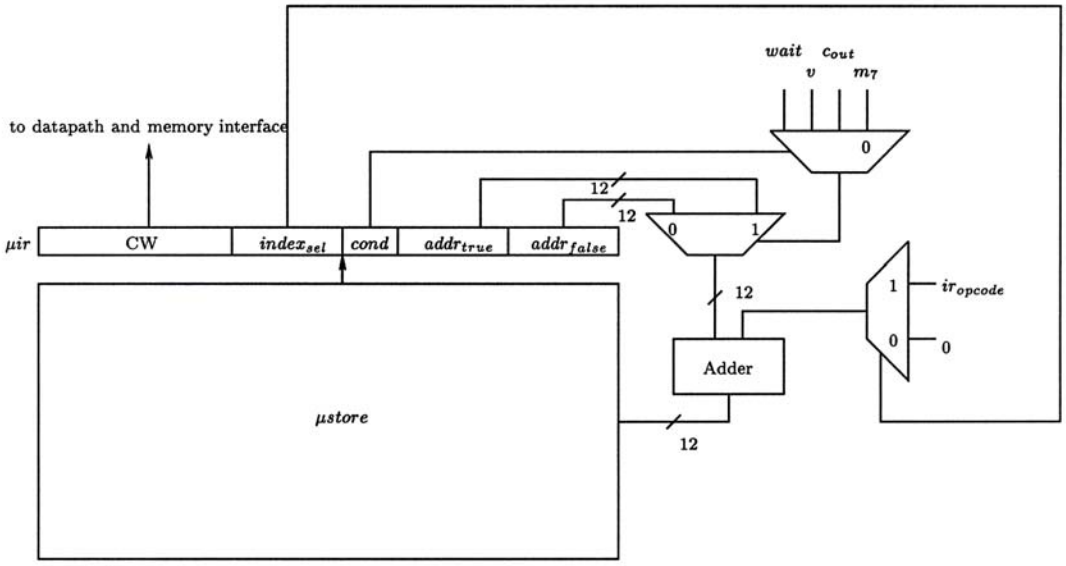
CS 366 - Computer Architecture II



CS 366 - Computer Architecture II

Decoding Instructions in a Microprogram

The final thing to add is how the opcode bits are taken from the instruction register and used by the control unit. The control unit uses the opcode value to determine the address of the next microinstruction to be executed. Once a base address is determined (this is done according to normal 2-way method: each microinstruction has two addresses of possible next microinstructions and specifies one status line. If the status line has a value of 1 the first address is used otherwise the second address is ised), the opcode value is added to the base address to obtain a new address. The addition of the opcode with the base address is only used if the current microinstruction has a certain value set to one. This value is the "indexsel" bit of the microinstruction. The following shows how this is done.



The microinstructions for the Mythsym machine specify the next microinstruction by the use of the keyword **goto**. When writing out the microinstructions will always use a label to specify each microinstruction instead of specifying an address. This allows us to rearrange the microinstructions without having to keep updating the address values. Once we are finished making changes to a microprogram, we would then translate the labels into their corresponding addresses. The following show how this is used. The labels will be given in italic font.

1. goto *fetch1*
2. if cout then goto *fetch0* else goto *branch*
3. goto *opcode*[iropcode]

The following will be the actual information stored in a microinstruction to be used by the next address generator if the above three examples were part of an actual microinstruction.

CS 366 - Computer Architecture II

Number	indexSel	cond	addrTrue	addrFalse
1	0	any	*fetch1*	*fetch1*
2	0	cout	*fetch0*	*branch*
3	1	any	*opcode*	*opcode*

In the first example, the next microinstruction to be executed will always be the microinstruction with label *fetch1*. So both addrTrue and addrFalse have the value of fetch1 and the cond can be any status line. The opcode value from the instruction register is not used here to the indexSel value is 0.

In the second example, the next microinstruction to be executed will be the microinstruction with the label *fetch0* if the cout status line is 1 (true) or the microinstruction with the label *branch* if the cout status line is 0 (false). The opcode value from the instruction register is not used here to the indexSel value is 0.

In the third example, the next instruction to be executed will be determined by the opcode value from the instruction regester. The opcode value will be added to the address of the microinstruction with label *opcode* to determine which microinstruction to execute next. The example shows the label *opcode* as an array to help specify this operation.

CS 366 - Computer Architecture II

Control Programs

A control program is section of code that will specify the control lines set inside of a control unit. The syntax of each line is a listing of the control lines being used and its value. Any control line not listed is assumed to have value of zero. If a control line can only have values of zero or one, listing the control line implies a value of one while not listing the control line implies a value of zero.

Let use create a tiny instruction set using the Mythsim architecture. We will have the following operations:

Name	**Operation**	**Opcode**
No-op	does nothing	0
Add	ri ← rj + rk	1
Set Register	ri ← const8	2
Branch if zero	if (rj == 0) PC ← PC + const4	3
Move	ri ← rj	4
Store	Mem[rj] ← rk	5
Load	ri ← Mem[rj]	6
Subtraction	ri ← rj - rk	7

The first part of the control program will be to fetch the next instruction based on the current value of the program counter. We saw this before, but we wish to redo it efficiently. We use register 6 to hold a copy of the PC that we can increment. This allows a more efficient way of doing the operation.

Address	**Label**	**Control Lines**
0	fetch0	asel = 7, bsel = 7, alusel = AND, r6write, marsel = LOAD, goto fetch1;
1	fetch1	asel = 6, alusel = ADDA, cin, r7write, read, ir0sel = LOAD, if wait then goto fetch1 else goto fetch2 endif;
2	fetch2	asel = 7, bsel = 7, alusel = AND, r6write, marsel = LOAD, goto fetch 3;
3	fetch3	asel = 6, alusel = ADDA, cin, r7write, read, ir1sel = LOAD, if wait then goto fetch3 else goto fetch4 endif;

The next part of the control program will be to decode the opcode. This is done by using a special input to determine the next instruction. This uses the opcode bits from the instruction register. Thus iropcode would be another status line.

Address	**Label**	**Control Lines**
4	fetch4	goto opcode[iropcode];

CS 366 - Computer Architecture II

Now we need to determine what to do for each opcode. We won't worry about the addess values until we combine everything together.

Opcode 0: No-op

This operation is to do nothing. This is often useful in synchronizing other operations. The control line would be:

Address	Label	Control Lines
	opcode[0]	goto fetch0;

Opcode 1: Add - ri ← rj + rk

As pointed out before, this operation relies on the values in the instruction register.

Address	Label	Control Lines
	opcode[1]	risel, rjsel, rksel, alusel = ADD, goto fetch0;

Opcode 2: Set Register - ri ← const8

This instruction will fill in a register with the value from the last 8 bits of the instruction register.

Address	Label	Control Lines
	opcode[2]	risel, resultsel = CONST8, goto fetch0;

Opcode 3: Branch if zero - if (rj == 0) then PC ← PC + const4

This instruction will check if a register value is zero. If so, it will add the 4 bit value from the instruction register to the PC. To check if a value is zero, we can use the decrement operation in the ALU (SUBA) and check the cout signal line. If cout is one, the value was not zero. If cout is zero, the value was zero.

Address	Label	Control Lines
	opcode[3]	rjsel, alusel = SUBA, r6write, resultsel = CONST4, if cout then goto fetch0 else goto branch endif;
	branch	r7write, asel = 7, bsel = 6, alusel = ADD, goto fetch0;

Opcode 4: Move - ri ← rj

This instruction is to move a value from one register to another. The use of the increment operation without setting the value of cin is used to move the value through the ALU unchanged.

Address	Label	Control Lines
	opcode[4]	risel, rjsel, alusel = ADDA, goto fetch0;

Opcode 5: Store - Mem[rj] ← rk

This instruction is to place a value from a register into the computer's memory. The address will be specified in another register. The use of the label memwrite is used to allow for other write operations (not yet defined). Note the two control statements that are needed to access the value to be written.

Address	**Label**	**Control Lines**
	opcode[5]	rjsel, alusel = ADDA, marsel = LOAD, goto opcode5a;
	opcode5a	asel = 5, bsel = 5, alusel = SUB, cin, r5write, goto opcode5b
	opcode5b	rksel, asel = 5, alusel = OR, mdrsel = LOAD_ALU, goto memwrite;
	memwrite	write, if wait then goto memwrite else goto fetch0 endif;

Opcode 6: Load - ri ← Mem[rj]

This instruction will place a value from the computer's memory into a register. The address will be specified in another register.

Address	**Label**	**Control Lines**
	opcode[6]	rjsel, alusel = ADDA, marsel = LOAD, goto opcode6a;
	opcode6a	read, mdrsel = LOAD_MEM, if wait then goto opcode6a else goto opcode6b endif;
	opcode6b	resultsel = MDR, risel, goto fetch0;

Opcode 7: Subtraction - ri ← rj - rk

This is similar to the addition operation, except with a different alu operation.

Address	**Label**	**Control Lines**
	opcode[7]	risel, rjsel, rksel, aulsel = SUB, cin, goto fetch0;

Putting it all together we get. Note that the opcode[X] labels are all listed sequentially.

Address	Label	Control Lines
0	fetch0	asel = 7, bsel = 7, alusel = AND, r6write, marsel = LOAD, goto fetch1;
1	fetch1	asel = 6, alusel = ADDA, cin, r7write, read, ir0sel = LOAD, if wait then goto fetch1 else goto fetch2 endif;
2	fetch2	asel = 7, bsel = 7, alusel = AND, r6write, marsel = LOAD, goto fetch 3;
3	fetch3	asel = 6, alusel = ADDA, cin, r7write, read, ir1sel = LOAD, if wait then goto fetch3 else goto fetch4 endif;
4	fetch4	goto opcode[iropcode];
5	opcode[0]	goto fetch0;
6	opcode[1]	risel, rjsel, rksel, alusel = ADD, goto fetch0;
7	opcode[2]	risel, resultsel = CONST8, goto fetch0;
8	opcode[3]	rjsel, alusel = SUBA, r6write, resultsel = CONST4, if cout then goto fetch0 else goto branch endif;
9	opcode[4]	risel, rjsel, alusel = ADDA, goto fetch0;
10	opcode[5]	rjsel, alusel = ADDA, marsel = LOAD, goto opcode5a;
11	opcode[6]	rjsel, alusel = ADDA, marsel = LOAD, goto opcode6a;
12	opcode[7]	risel, rjsel, rksel, aulsel = SUB, cin, goto fetch0;
13	branch	r7write, asel = 7, bsel = 6, alusel = ADD, goto fetch0;
14	opcode5a	asel = 5, bsel = 5, alusel = SUB, cin, r5write, goto opcode5b
15	opcode5b	rksel, asel = 5, alusel = OR, mdrsel = LOAD_ALU, goto memwrite;
16	memwrite	write, if wait then goto memwrite else goto fetch0 endif;
17	opcode6a	read, mdrsel = LOAD_MEM, if wait then goto opcode6a else goto opcode6b endif;
18	opcode6b	resultsel = MDR, risel, goto fetch0;