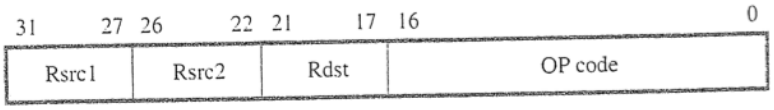
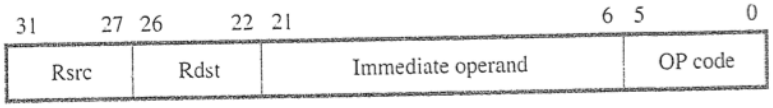
CS 317: Computer Organization
Semester Project

1 Overview

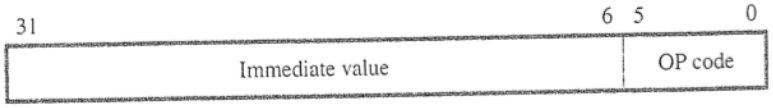
Your assignment will be to design, describe, and build a simple processor *similar to* that discussed in Chapter 5 of your textbook. We will all be using the same datapath, shown in Figure 5.18 and related figures of your textbook and at the end of this packet and variations on the IR fields shown below (and in Figure



(a) Register-operand format



(b) Immediate-operand format



(c) Call format

Figure 5.12 Instruction encoding.

5.12 of your text). This datapath will be jointly developed by the class as Phase 1 of this project and then shared by all project teams, as discussed below.

All processor busses and registers should be 32 bits wide, and you should have 32 (nominally) general purpose registers, numbered R0-R31. All of the registers shown in the datapath are also 32-bit registers. This implies that you will need five bits to specify each of the register fields, as shown above. We will use variations of the textbook formats when implementing our processor and your assigned portion(s) of the datapath.

2 Phase 1

The four person teams were previously required to design and provide various portions of the datapath and/or debugging/display/helper functionality. Of the 20% of the class grade dedicated to "Projects, Demos, and Reports", this will counts as 1/5 (or 4% of your overall grade). Your team must provide required

functionality (matching the interfaces specified and signal names implied in the textbook, Chapter 5, unless otherwise specified) and documentation.

All groups are required to use this common functionality in the later phases of this project. If a module is not correctly implemented or it's use in unclear, any member of any other group can ask that the module be repaired. If the class reaches consensus that a module is not correctly implemented and the original team refuses to make the changes, other groups can make the modifications and thereby earn a portion of the original groups 4% (at the expense of the original group). The rest of the class must agree that the original module was unsuitable and that the replacement is suitable. If consensus cannot be reached, I will decide but I want you guys to do this without me as much as possible. It should go without saying (but I will say it) that Dr. Fourney can override "the class" on any of these decisions, but I hope to not have to do so. If I tell you how to do it, you are not "designing" and I am sure that you are all mature enough that I won't have to step in to settle petty squabbles.

While this portion was due a couple of weeks ago, but apparently the class has not reached consensus on many of these modules. It is unclear if this is because the modules are not correct or because the class is slow to get started on Phase 2 of the project, which was initially to be demonstrated on Monday, November 3rd. In either case, the groups are not communicating very well. Please use the discussion page extensively between now and Friday, and we will use Friday's class (as much as is needed) to finally resolve any of these issues. For a hypothetical example, if people are having trouble using the ROM, or implementing an *.mif file, the group that designed that module could (and should) provide the Verilog test module to read a word from that ROM module and/or a simple MIF file with a couple of Phase 2 instructions. If these hypothetical situations get resolved before Friday, wonderful. If not, we will decide on Friday who can provide a functional (again, hypothetical) ROM module and steal some of this groups points.

No matter how the discussion turns out this week, come to class with everything you need to convince us your modules are working. We do not want a flashy slide show, we want you to show us how your module works and convince us that it does work.

While this should technically be the end of Phase 1, you may very well find that, for example, some Phase 3 operations exercise the datapath in unanticipated ways. When (not if) these situations occur any group can propose a change to the module in question. Ideally these changes should be agreed upon by the other groups and then updated. Any such changes should obviously not break any earlier functionality. E.g., a change to the address generation unit which is required to implement a Phase 3 operation should not break any Phase 2 operations and should require minimal additions to the interface–for example, one additional select line for a mux may need to be added, but the other control lines should be unchanged so long as the new line is in one state but may provide new and different functionality when this control line is in the other state.

DRAFT COPY

3 Phase 2

This phase deals with the list of opcodes that were previously due on Monday, November 3rd and have now been extended until Monday, November 10th. These must be implemented using the common datapath. Each team (of two) is write their own control unit and their own ALU, but these components should drop right into the datapath everybody else is using and make use of the provided helper/debug functionality. This phase is worth 40% of your project grade (so 8% of your overall grade for the course). If you need to develop some of the Phase 1 modules on your own to complete your work in a timely manner that could be used as evidence to request changes, but your project must be demonstrated with the common group modules. There may or may not be a separate written report (due after the demo) on these Phase 2 Opcodes. If required, that report will fall under the 8% mentioned above.

Subject to our discussion in class on Friday, October 17th, you will support following functions and the corresponding instruction word formats for Phase 2 of this project. These are mostly "Format (a)" instructions, with a couple of immediate loads ("Format (b)") thrown in to help with testing, debugging, and development.

I've sorted these by which IR format (See Figure 5.12 and above) is most relevant, and given on the first line the opcode, a mnemonic to refer to the instruction by, and an optional phrase the mnemonic stands for if I don't think it's obvious. This is followed by a brief description where required.

bf Phase 2 OpCodes

```
Variations on Format (a)

XXXX_XXXXXXX_111111     NOP   No OPeration
      Do nothing (but take all five cycles to do it) Note that this is any
      instruction with the 6 least significant bits equal to 111111, so
      this pattern can not be used for formats (b) or (c) either.

0000_0000001_000000 ADD
       The contents of Rsrc1 and Rsrc2 are added, the result is stored in Rdst

0000_0000100_000000   SUB
       The contents of Rsrc2 are subtracted from the contents of Rsrc1 and
       the result is stored in Rdst

0000_0001100_000000   COMP
       The bitwise (one's) complement of the contents of Rsrc1 are placed in Rdst

0000_0001010_000000   NEG
       The two's complement of the contents of Rsrc1 are placed in Rdst

0000_0001000_000000   AND
```

DRAFT COPY

The contents of the two source registers are logically ANDed together (bitwise AND) and the result is placed in Rdst

0000_0001001_000000 OR
The contents of the two source registers are logically ORed together (bitwise OR) and the result is placed in Rdst

0000_0001011_000000 XOR The contents of the two source registers are logically XORed together (bitwise XOR) and the result is placed in Rdst

0000_0010011_000000 ASL Arithmetic Shift Left (shift one bit position only)
This is a 33 bit shift of the contents of Rsrc1 with the left-most bit of Rsrc1 going to the Carry flag and a 0 being shifted into the right-most bit position of Rdst

0000_0010001_000000 ASR Arithmetic Shift Right (shift one bit position only)
This is a 33 bit shift of Rsrc1 with the right-most bit of Rsrc1 going to the Carry flag and a sign extension used to fill left-most bit position of Rdst

0000_0010011_000000 LSL Logical Shift Left (shift one bit position only)
This is a 33 bit shift of Rsrc1 with the left-most bit of Rsrc1 going to the Carry flag and a 0 being shifted into the right-most bit position of Rdst

0000_0010000_000000 LSR Logical Shift Right (shift one bit position only)
This is a 33 bit shift of Rsrc1 with the right-most bit of Rsrc1 going to the Carry flag and a 0 being shifted into the left-most bit position of Rdst

0000_0011010_000000 ROL Rotate Left (by one bit position)
Affects carry bit in the same manner as the shifts

0000_0011001_000000 ROR Rotate Right (by one bit position)
Affects carry bit in the same manner as the shifts

0000_0100000_000000 MOVE (Copy)
The contents of Rsrc1 are copied into Rdst

0000_0100001_000000 LBI Load Base with Index
The unsigned contents of the two source registers are added to create the EA. Rdst is then loaded from memory location EA

0000_0100010_000000 LDRi Load Register Indirect
Rsrc2 contains a pointer to the value to be copied into Rdst

Variations on Format (b)

100010 LD # Load immediate

DRAFT COPY

```
    The value in the immediate field is sign extended and placed in the Rdst.

100011 LDU Rx #num    Load unsigned immediate
    The value in the immediate field is padded with zeros to the left and
    placed in Rdst
```

4 Phase 3

These are the additional opcodes (branches, jumps, indexed loads, stores, subroutine calls, etc) mentioned in class but deferred to the later phase. These will need to be implemented and demonstrated in early December. A report will be required (either covering just Phase 3, or all three phases, depending on the reporting requirements for Phase 2). These instructions would also require the use of the common datapath, which will have to be slightly modified in the manner discussed in class previously. Regardless of required modifications, the same modified datapath must be used by all teams.

This portion will constitute the remaining 40% of your project grade (8% of the course grade). If only a single report is required, the report grade will be divided and incorporated into the 8% for both the Phase 2 and the Phase 3 grades.

I had initially planned on, and am still leaning towards, shuffling the partners/teams for Phase 3, but am willing to entertain suggestions on this.

Phase 3 OpCodes

```
Variations on Format (a)

0000_1000000_000000   JMP   Jump
     Place the contents of Rsrc1 into the PC

0000_1000001_000000   JSR   Jump to Subroutine
     Address of subroutine in Register Rsrc1, store return address in LINK
     register, which is always R30.

0000_1000011_000000   RTS  Return from Subroutine
    Rsrc1 contains the register number for the link register (R30).

Variations on Format (b)

000001 ADD #
    The immediate value is sign extended and added to the contents of Rsrc.  The
    result is stored in Rdst

000100 SUB #
    The immediate value is sign extended and subtracted from the contents of
    Rsrc.  The result is stored in Rdst
```

DRAFT COPY

001000 AND #

The immediate value is padded with zeros on the left and ANDed with the contents of Rsrc. The result is place in Rdst

001001 OR #

The immediate value is padded with zeros on the left and ORed with the contents of Rsrc. The result is place in Rdst

001011 XOR #

The immediate value is padded with zeros on the left and XORed with the contents of Rsrc. The result is place in Rdst

001100 BEQ Branch if EQual

If the contents of the two registers are equal, add the 2's complement immediate value to the PC

001010 BNE Branch if Not Equal

If the contents of the two registers are not equal, add the 2's complement immediate value to the PC

001111 BLT Branch if Less Than

If the unsigned contents of Rsrc are less than the contents of Rdst, add the 2's complement immediate value to the PC

100000 LDA Load Absolute

The immediate value is zero-filled to the left and used as an address. Rdst is then loaded from this address. This requires the adder in Figure 5.10 to be able to just pass the immediate value through (without adding to the PC), which requires an additional control line that is not implied by Figure 5.10

010000 STA STore Absolute

The immediate value is zero-filled to the left and used as an address. Rdst (yes, Rdst!) is then stored to this address. Requires modifications similar to the LDA instruction

100001 LDIX LoaD IndeXed

The unsigned immediate value is added to the contents of Rsrc to obtain the EA. Rdst is then loaded from the memory location EA

010001 STIX STore IndeXed

The unsigned immediate value is added to the contents of Rsrc to obtain the EA. Rdst is then stored to the memory location EA

DRAFT COPY

Variations on Format (c)

110000 BRA Branch
Add the 2's complement immediate value to the PC.

110001 BSR Branch to SubRoutine
Add the 2's complement immediate value to the PC and store return address in the LINK register, which is always R30.

DRAFT COPY

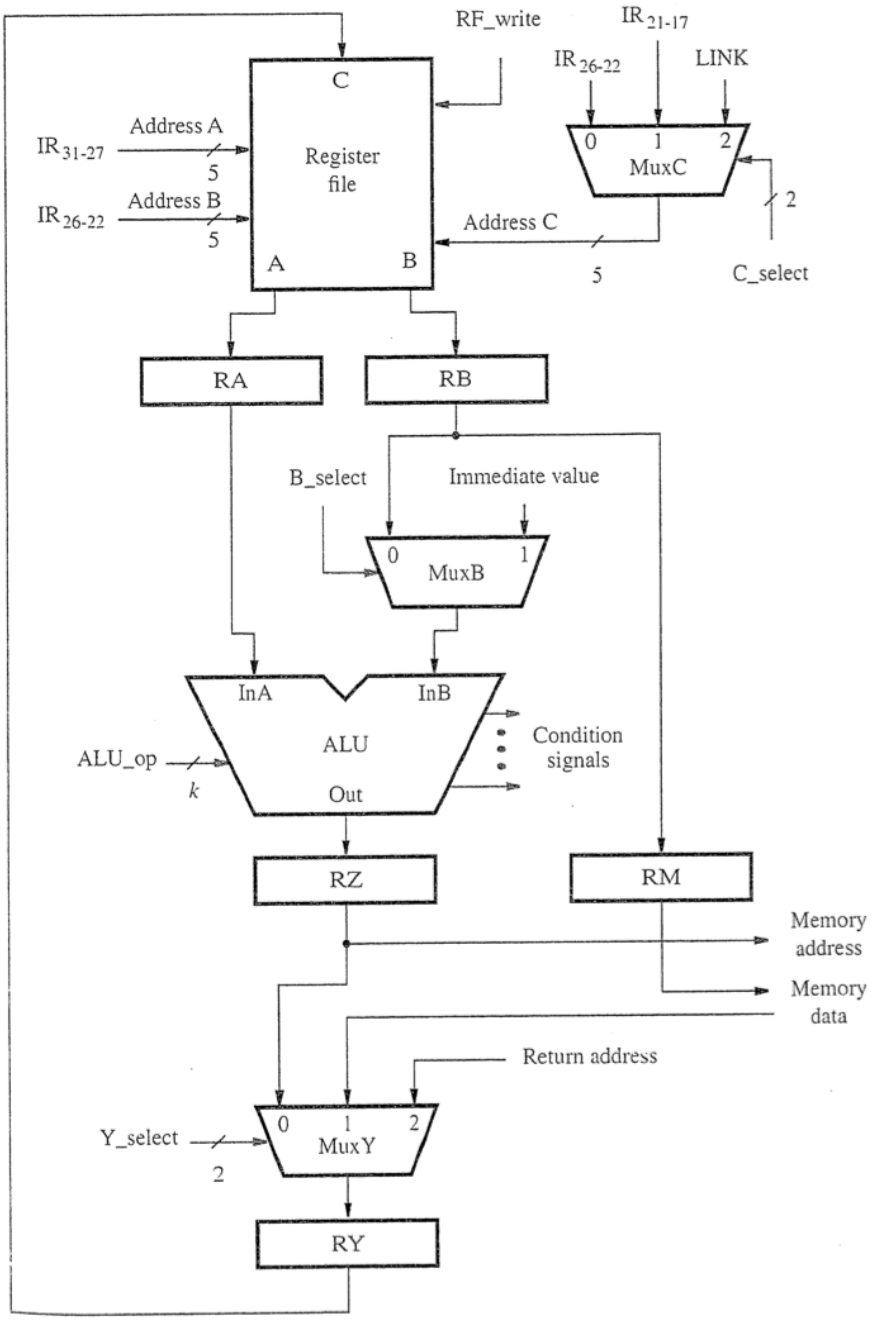


Figure 5.18 Control signals for the datapath.