UNIVERSITY of York

Department of Electronics

Computer Architectures

Homework One

Abstract

Homework one for Computer Architectures module.

Y3839090

March 8, 2017

Contents

List of Figures

1 Question 1

1.1 Extra Instructions

The current instruction set for architecture B is lacking the XOR and AND instructions. These are both logic instructions so they will be grouped with the other logic instructions. This means there opcodes will start with 01 (the same as the other logic instructions). Both of these instruction take the three operands Rt, Ra and Rb and so will use the same instruction coding as the ADD and SUB instructions (ADD and SUB also take Rt, Ra and Rb operands).

Table 1: The extra instructions need for Arch B to run the program

Command	Operands	Opcode	Fields
and	Rt, Ra, Rb	01 100	xxx xxxx xBBB xxxx xAAA xxxx xTTT
xor	Rt, Ra, Rb	01 110	xxx xxxx xBBB xxxx xAAA xxxx xTTT

1.2 Conversions

1.2.1 Assembly

This is the given pseudo code converted to the assembly language defined for architecture B.

Assembly code

```
    ; data_addr is in R7
    ; word_length is in R1

    move R2, R7                 ; R2 is data
    movi R3, 0x0000             ; R3 is parity
    move R4, R1                 ; R4 is i
    movi R5, 0x0001             ; R4 is mask

loop_start_label:
    br R4, 000, loop_end_label      ; if i is zero exit the loop
    and R6, R2, R4              ; bit mask data
    xor R3, R6, R3              ; keep track of parity
    shr R2, R2, 1               ; go to next bit
    dec R4, R4
    br R0, 000, loop_start_label    ; go back to the beginning of the
   loop

loop_end_label:
    storo R7, R3, 2          ; store output
```

1.2.2 Machine code - Binary

This is the above assembly code hand assembled into binary. “x" has been used to represent “don’t care” bits.

Machine code - Binary with don’t cares

```
1  1000 1xxx  xxxx xxxx    xxxx x111  xxxx x010 ; move R2, R7
2  1000 0xxx  0000 0000    0000 0000  xxxx x011 ; movi R3, 0x0000
3  1000 1xxx  xxxx xxxx    xxxx x001  xxxx x100 ; move R4, R1
4  1000 0xxx  0000 0000    0000 0001  xxxx x011 ; movi R5, 0x0001
5                                               ; loop_start_label:
6  1100 0000  0000 0110    xxxx x100  xxxx xxxx ; brz R4, loop_end_label
7  0110 0xxx  xxxx x100    xxxx x010  xxxx x110 ; and R6, R2, R4
8  0111 0xxx  xxxx x011    xxxx x110  xxxx x011 ; xor R3, R6, R3
9  0100 1xxx  xxxx 0001    xxxx x010  xxxx x010 ; shr R2, R2, 1
10 0001 1xxx  xxxx xxxx    xxxx x100  xxxx x100 ; dec R4, R4
11 1100 0111  1111 1011    xxxx x000  xxxx xxxx ; br R0, 000,
       loop_start_label
12                                              ; loop_end_label:
13 1011 1000  0000 0010    xxxx x011  xxxx x111 ; storo R7, R3, 2
```

To be able to convert this to hex, the “don’t care” bits have to changed to either 0 or 1. I decided to convert all of the “dont care” bits to 0’s.

Machine code - Binary, don’t cares replace with 0s

```
1  1000 1000  0000 0000    0000 0111  0000 0010 ; move R2, R7
2  1000 0000  0000 0000    0000 0000  0000 0011 ; movi R3, 000000
3  1000 1000  0000 0000    0000 0001  0000 0100 ; move R4, R1
4  1000 0000  0000 0000    0000 0001  0000 0011 ; movi R5, 000001
5                                               ; loop_start_label:
6  1100 0000  0000 0110    0000 0100  0000 0000 ; brz R4, loop_end_label
7  0110 0000  0000 0100    0000 0010  0000 0110 ; and R6, R2, R4
8  0111 0000  0000 0011    0000 0110  0000 0011 ; 0or R3, R6, R3
9  0100 1000  0000 0001    0000 0010  0000 0010 ; shr R2, R2, 1
10 0001 1000  0000 0000    0000 0100  0000 0100 ; dec R4, R4
11 1100 0111  1111 1011    0000 0000  0000 0000 ; jmp loop_start_label
12                                              ; loop_end_label:
13 1011 1000  0000 0010    0000 0011  0000 0111 ; storo R7, R3, 2
```

1.2.3 Machine code - Hex

This is the above binary machine code shown as hexadecimal. Remember all “don’t cares” have been converted to zero so that the values can be represented in hex.

Machine code - Hex, don’t cares replace with 0s

```
1  0x88000702   ; move R2, R7
```

```
0x80000003  ; movi R3, 000
0x88000104  ; move R4, R1
0x80000103  ; movi R5, 001
            ; loop_start_label:
0xC0060400  ; brz R4, loop_end_label
0x60040206  ; and R6, R2, R4
0x70030603  ; 0or R3, R6, R3
0x48010202  ; shr R2, R2, 1
0x18000404  ; dec R4, R4
0xC7FB0000  ; jmp loop_start_label
            ; loop_end_label:
0xB8020307  ; storo R7, R3, 2
```

1.3 Control Signals

Command	Opcode	RA[2:0]	RB[2:0]	WA[2:0]	IMM[15:0]	OEN	S[4:1]	Al[2:0]	SH[5:0]	WEN
move R2, R7	10001	111	000	010	0xΦΦΦΦ	0	0ΦΦ0	101	ΦΦ0000	0
movi R3, 0x0000	10000	000	ΦΦΦ	011	0x0000	0	0ΦΦ1	101	ΦΦ0000	0
move R4, R1	10001	001	000	001	0xΦΦΦΦ	0	0ΦΦ0	101	ΦΦ0000	0
movi R5, 0x0001	10000	000	ΦΦΦ	101	0x0001	0	0ΦΦ1	101	ΦΦ0000	0
br R4, 000, loop_end_label	11000	100	000	ΦΦΦ	0x0005	0	ΦΦΦ1	101	ΦΦ0000	0
and R6, R2, R4	01100	010	100	110	0xΦΦΦΦ	0	0ΦΦ0	001	ΦΦ0000	0
xor R3, R6, R4	01110	110	100	011	0xΦΦΦΦ	0	0ΦΦ0	010	ΦΦ0000	0
shr R2, R2, 1	01001	010	000	010	0xΦΦΦΦ	0	0ΦΦ0	101	100001	0
dec R4, R4	00011	100	ΦΦΦ	100	0xΦΦΦΦ	0	0ΦΦ0	100	ΦΦ0000	0
br R0, 000, loop_start_label	11000	ΦΦΦ	000	ΦΦΦ	0xFFFB	0	ΦΦΦ1	101	ΦΦ0000	0
storo R7, R3, 2	10111	011	ΦΦΦ	111	0x0002	1	00Φ1	101	ΦΦ0000	1

Figure 1: Stages and control signals

2 Question 2

Here is a table showing all the control signals need to execute the instruction with the multi-cycle architecture C.

3 Question 3

Here is my encoding for each of the instructions in the set.

Command	Steps	RA[2:0]	RB[2:0]	WA[2:0]	MA[15:0]	IMM[15:0]	OEN	S[1:4]	AL[2:0]	SH[5:0]	WEN
SHR R3, R1, 5	S1 - Fetch	ØØØ	ØØØ	ØØØ	0xØØØØ	0xØØØØ	0	Ø1ØØ	111	ØØØØØ	0
	S2 - Reg R	000	001	ØØØ	0xØØØØ	0xØØØØ	0	ØØØØ	ØØØ	ØØØØØ	0
	S3 - ALU	ØØØ	ØØØ	ØØØ	0xØØØØ	0xØØØØ	0	0ØØ0	011	10101	0
	S4 - Reg W	ØØØ	ØØØ	011	0xØØØØ	0xØØØØ	0	ØØØ0	ØØØ	ØØØØØ	1
LOADI R5, 0xAF1F	S1 - Fetch	ØØØ	ØØØ	ØØØ	0xØØØØ	0xØØØØ	0	Ø1ØØ	111	ØØØØØ	0
	S4 - Mem RW	ØØØ	ØØØ	ØØØ	0xAF1F	0xØØØØ	0	ØØ1Ø	ØØØ	ØØØØØ	0
	S5 - Reg W	ØØØ	ØØØ	101	0xØØØØ	0xØØØØ	0	ØØØ1	ØØØ	ØØØØØ	1
BRNEQ R3, 0x11A	S1 - Fetch	ØØØ	ØØØ	ØØØ	0xØØØØ	0xØØØØ	0	Ø1ØØ	111	ØØØØØ	0
	S2 - Reg R	011	000	ØØØ	0xØØØØ	0x011A	0	11ØØ	101	ØØØØØ	0
	S3 - ALU	ØØØ	ØØØ	ØØØ	0xØØØØ	0xØØØØ	0	0ØØ0	101	ØØØØØ	0

Figure 2: Stages and control signals

Category	Instruction	Operands	Operation	Opcode	Fields
No-operation	nop	N/A	none	00 00 00	ΦΦ ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ
Arithmetic	add	rt,ra,rb	rt <= ra + rb	00 10 00	ΦΦ ΦΦΦΦ ΦΦΦΦ BBBB BAAA AAΦT TTTT
	sub	rt,ra,rb	rt <= ra – rb	00 01 00	ΦΦ ΦΦΦΦ ΦΦΦΦ BBBB BAAA AAΦT TTTT
	addi	rt,ra,imm	rt <= ra + immediate value	00 10 01	II IIII IIII IIII ΦAAA AAΦT TTTT
	subi	rt,ra,imm	rt <= ra – immediate value	00 01 01	II IIII IIII IIII ΦAAA AAΦT TTTT
	inc	rt,ra	rt <= ra + 1	00 10 10	ΦΦ ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ ΦAAA AAΦT TTTT
	dec	rt,ra	rt <= ra – 1	00 01 10	ΦΦ ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ ΦAAA AAΦT TTTT
Logic	not	rt,ra	rt <= NOT ra	01 11 11	ΦΦ ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ ΦAAA AAΦT TTTT
	and	rt,ra,rb	rt <= ra AND rb	01 01 00	ΦΦ ΦΦΦΦ ΦΦΦΦ BBBB BAAA AAΦT TTTT
	or	rt,ra,rb	rt <= ra OR rb	01 10 00	ΦΦ ΦΦΦΦ ΦΦΦΦ BBBB BAAA AAΦT TTTT
	xor	rt,ra,rb	rt <= ra XOR rb	01 11 00	ΦΦ ΦΦΦΦ ΦΦΦΦ BBBB BAAA AAΦT TTTT
	andi	rt,ra,imm	m rt <= ra AND immediate value	01 01 01	II IIII IIII IIII ΦAAA AAΦT TTTT
	ori	rt,ra,imm	rt <= ra OR immediate value	01 10 01	II IIII IIII IIII ΦAAA AAΦT TTTT
	xori	rt,ra,imm	rt <= ra XOR immediate value	01 11 01	II IIII IIII IIII ΦAAA AAΦT TTTT
	shl	rt,ra,n	rt <= ra shifted left by n bits	01 00 01	ΦΦ NNNN ΦΦΦΦ ΦΦΦΦ ΦAAA AAΦT TTTT
	shr	rt,ra,n	rt <= ra shifted right by n bits	01 00 00	ΦΦ NNNN ΦΦΦΦ ΦΦΦΦ ΦAAA AAΦT TTTT
	rol	rt,ra,n	n rt <= ra rotated left by n bits	01 00 11	ΦΦ NNNN ΦΦΦΦ ΦΦΦΦ ΦAAA AAΦT TTTT
	ror	rt,ra,n	rt <= ra rotated right by n bits	01 00 10	ΦΦ NNNN ΦΦΦΦ ΦΦΦΦ ΦAAA AAΦT TTTT
Transfer	move	rt,ra	rt <= ra	10 00 00	ΦΦ ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ ΦAAA AAΦT TTTT
	loadi	rt,addr	rt <= DMEM[addr] {direct addressing}	10 01 01	AA AAAA AAAA AAAA AAAA AAΦT TTTT
	loadr	rt,ra	rt <= DMEM[ra] {register indirect addressing}	10 01 10	ΦΦ ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ ΦAAA AAΦT TTTT
	loado	rt,ra,off	rt <= DMEM[ra+off] {base plus offset addressing}	10 01 11	ΦΦ ΦΦΦΦ OOOO OOOO ΦAAA AAΦT TTTT
	stori	rt,addr	DMEM[addr] <= rb {direct addressing}	10 10 01	AA AAAA AAAA AAAA AAAA AAΦT TTTT
	storr	rt,ra	DMEM[ra] <= rb {register indirect addressing}	10 10 10	ΦΦ ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ ΦAAA AAΦT TTTT
	storo	rt,ra,off	DMEM[ra+off] <= rb {base plus offset addressing}	10 10 11	ΦΦ ΦΦΦΦ OOOO OOOO ΦAAA AAΦT TTTT
control	jmp	off	Jump to IMEM[PC+off]	11 00 00	ΦΦ ΦΦΦΦ OOOO OOOO ΦΦΦΦ ΦΦΦΦ ΦΦΦΦ
	brc	ra,cond,off	f If condition is true, then jump to IMEM[PC+off], else continue Conditions: ra = 0 ; ra ≠ 0 ; ra = 1; ra < 0; ra > 0; ra ≤ 0; ra ≥ 0	11 10 00	ΦΦ ΦΦΦΦ OOOO OOOO ΦAAA AAΦΦ ΦCCC

Figure 3: Instruction set encoding