Lab1

Tuesday, January 18, 2022 7:16 PM

Prelab1

(1) [1 point]

Write the instruction to add the contents of the r0 register to the contant value 0x88 and write the result to the r0 register. **Specify the instruction using syntax you would use for an assembler. Make sure you use a '#' character to specify the immediate value. Try it in the simulator to make sure it assembles.**

adds r0,r0,#0x88

Save

---

(2) [1 point]

Write the instruction to add the contents of the r7 register to the constant value 4 and write the result to the r6 register. **Specify the instruction using syntax you would use for an assembler. Make sure you use a '#' character to specify the immediate value. Try it in the simulator to make sure it assembles.**

adds r6,r7,#4

Save

---

(3) [1 point]

For the instruction you specified in (2), if you changed the immediate constant, what is the largest value you could use? Check this in the simulator by trying a value one larger than what you think is highest. Specify this as a decimal number.

7

Save

---

(4) [1 point]

cmp r3,r4 ⇒ 0100 0010 1011 0011
⇒ 0100 0010 1010 0011
⇒ 4 2 a 3

Look at the ARMv6-M Architecture Reference Manual, section A6.7.2 and write the encoding for the instruction that you listed in question (2). Express your answer as a four-digit hexadecimal number.

adds r6,r7,#4
imm3 Rn Rd
0001110 | 100 | 111 | 110
1 C 3 E

1C3E

Save

(5) [1 point]

There is no instruction to multiply the value of a register by a constant. Write a two-instruction sequence that multiplies the value of register r5 by 0xd6 and writes the result to register r6. Make sure that the only register modified by these instructions is r6. **Specify the instructions using syntax you would use for an assembler. Try them in the simulator to make sure they assemble.**

1st instruction: movs r6,#0xd6
2nd instruction: muls r6,r5

Save

---

(6) [1 point]

Write a multi-instruction sequence that adds the constant 0x8d to r2 and then multipies it by the contents of r6. In other words, r2 = (r2 + 0x8d) * r6. Do not modify any register other than r2 with these instructions. **Specify the instructions using syntax you would use for an assembler. Try them in the simulator to make sure they assemble.**

1st instruction: adds r2 #0x8d
2nd instruction: muls r2,r6

Save

---

(7) [1 point]

What is the maximum immediate value that can be "mov"ed into a register?
E.g., for the instruction "movs r0,#***n***" what is the largest number that you can write for ***n***? Specify a decimal number.

255

Save

---

(8) [1 point]

Look at the ARMv6-M Architecture Reference Manual, section A6.7.39 and write the encoding for the instruction **`movs r2,#0x55`**. Express your answer as a four-digit hexadecimal number.

2255

0010 0010 0101 0101
2 2 5 5

Save

(9) [1 point]

What is the instruction encoding of an instrution that adds the value 0x4a to register r4 and deposits the result in register r4? Express your answer as a four-digit hexadecimal value. (You can and should use the simulator or System Workbench to verify your result, but you need to know how to look this up in the appropriate documentation and determine the value manually for this and any other instruction.)

adds r4, #0x4a

3449

0011 0100 0100 1010
3 4 4 a

Save

---

(10) [1 point]

Write the assembly language specification for the instruction represented by the 16-bit hexadecimal code 36d8. In other words, write the assembly language phrase that will cause the assembler to create the instruction with that hexadecimal code. (You can and should use the simulator or System Workbench to verify your result, but you need to know how to look this up in the appropriate documentation and determine the value manually for this and any other instruction.)

adds R6, #0xd8

Save

0011 0110 1101 1000
00110 / Rdn / imm8
110 11011000
adds R6 #0xd8

Lab implementation details

You will implement several *subroutines* in this lab that will calculate the values of expressions that are uniquely specified for you. The expressions with example initial values and expected results are shown below. You should use these values to set up code you write for the lab.

step31:	
Expression	**`r0 = ((r3 - r0) * (r1 + r2))`**
Example inputs	r0 = -77; r1 = 40; r2 = -89; r3 = -32
Example result	-2205
step32:	
Expression	**`r0 = ((r1 + -r2) * (r3 - r0))`**
Example inputs	r0 = -53; r1 = -36; r2 = -52; r3 = 25
Example result	1248
step33:	
Expression	**`r0 = ((r1 - r3) + (r2 - r0))`**
Example inputs	r0 = 84; r1 = 98; r2 = 97; r3 = 83
Example result	28
step41:	
Expression	**`r0 = ((r3 | r0) ^ (r1 & r2))`**
Example inputs	r0 = 62; r1 = -76; r2 = 33; r3 = 57
Example result	31
step42:	
Expression	**`r0 = ((r0 & 0xb6) + (r1 & ~0x1f))`**
Example inputs	r0 = 53; r1 = -15; r2 = 26; r3 = -36
Example result	20

A6.7.2 ADD (immediate)

This instruction adds an immediate value to a register value, and writes the result to the destination register. It updates the condition flags based on the result.

Encoding T1 All versions of the Thumb instruction set.
`ADDS <Rd>,<Rn>,#<imm3>`

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	imm3			Rn			Rd		

```
d = UInt(Rd);  n = UInt(Rn);  setflags = !InITBlock();  imm32 = ZeroExtend(imm3, 32);
```

Encoding T2 All versions of the Thumb instruction set.
`ADDS <Rdn>,#<imm8>`

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	Rdn			imm8							

```
d = UInt(Rdn);  n = UInt(Rdn);  setflags = !InITBlock();  imm32 = ZeroExtend(imm8, 32);
```

A6.7.39 MOV (immediate)

Move (immediate) writes an immediate value to the destination register. The condition flags are updated based on the result.

Encoding T1 All versions of the Thumb instruction set.
`MOVS <Rd>,#<imm8>`

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	0	0	Rd			imm8							

```
d = UInt(Rd);  setflags = !InITBlock();  imm32 = ZeroExtend(imm8, 32);  carry = APSR.C;
```

A6.7.2 ADD (immediate)

This instruction adds an immediate value to a register value, and writes the result to the destination register. It updates the condition flags based on the result.

Encoding T1 All versions of the Thumb instruction set.
`ADDS <Rd>,<Rn>,#<imm3>`

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	1	1	1	0	imm3			Rn			Rd		

```
d = UInt(Rd);  n = UInt(Rn);  setflags = !InITBlock();  imm32 = ZeroExtend(imm3, 32);
```

Encoding T2 All versions of the Thumb instruction set.
`ADDS <Rdn>,#<imm8>`

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	1	1	0	Rdn			imm8							

```
d = UInt(Rdn);  n = UInt(Rdn);  setflags = !InITBlock();  imm32 = ZeroExtend(imm8, 32);
```