Midterm 1 Review - ANSWERS

EECS 370 – Introduction to Computer Organization - Winter 2016

Profs. Valeria Bertacco & Reetu Das

EECS Department
University of Michigan in Ann Arbor, USA

Assembly – W03Q10

What does this sequence of LC2K instructions do? What sequence of two ARM instructions could be used to replace this sequence ?

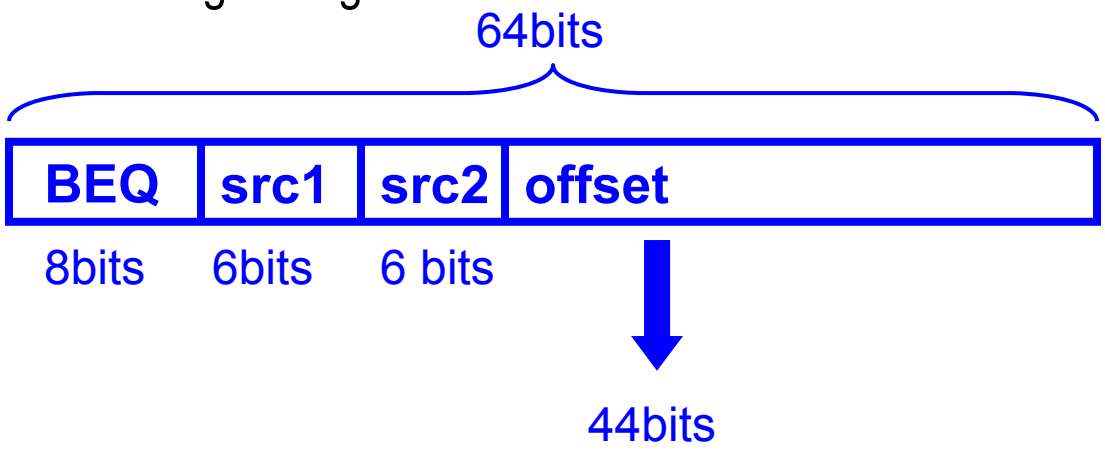
`add 1 1 2`	**`$2 = $1 + $1`**	**`$2 = 2*$1`**
`add 2 2 2`	**`$2 = $2 + $2`**	**`$2 = 4*$1`**
`add 1 2 2`	**`$2 = $2 + $1`**	**`$2 = 5*$1`**
`add 2 2 2`	**`$2 = $2 + $2`**	**`$2 = 10*$1`**

Ans:

```
mov r3, #10
mul r2, r1, r3          ; note: mul does not support an
                        ; imm for its second operand
```

ISA – W05Q1

Assume a 64-bit wide RISC machine with 64 registers and 200 distinct instructions. Its branch instruction compares two registers for equality, and uses the 2's complement offset field to compute the PC-relative target. What is the branch target range?



Nominal Ans: range -2^{43} to $2^{43}-1$

But branches really go to PC+offset+1 so range is more precisely: -2^43+1 to 2^43. Either answer is acceptable.

Addressing – F07 exam

Consider the following ARM assembly program:

```
ldrb   r4, [r0,#100]
strh   r4, [r4,#-42]
ldrsh  r3, [r1,#2]
str    r3, [r0,#101]
```

Initial value for Register 1 is:
r1 = 0x0000 0064
r0 = 0

Fill in the tables.

Reg	Initial value	After inst 1	After inst 2	After inst 3	Final values
r3	0x0000 0020			0xFFFF C700	
r4	0x0000 0074	0x0000 0091			

Address	Initial	After inst 1	After inst 2	After inst 3	Final values
100_{10}	0x91				
101_{10}	0x34				0xFF
102_{10}	0xC7				0xFF
103_{10}	0x84		0x00		0xC7
104_{10}	0x57		0x91		0x00
105_{10}	0xB3				

Memory layout – F05 exam (adapted)

- How many bytes does the C data structure below require?

```
struct foo {
    char a;
    int b, c;
    char d[10];
};
```

1byte + 3 (for alignment)
4 + 4
10
2 (struct as to be a multiple of 4)
---
24 bytes

- How can you rewrite the struct declaration so that it uses less memory ?

```
struct foo {
    int b, c;
    char d[10];
    char a;
};
```

20 bytes

Linker – F05 exam

file1.c

```
1: int c;
2: void foo(int a) {
3: int b;
4: reference to b
5: reference to c
6: bar(a);
7: }
```

file2.c

```
8: void bar(int x) {
9: int y[5];
10: reference to y
11: reference to x
12: reference to c
13: }
```

- What symbols will appear in the symbol tables of the two files ?
- What line numbers will be referenced in the two relocation tables ?

Symbol table: c, foo, bar
Reloc. table: 5, 6

Symbol table: c, bar
Reloc. table: 12

Caller/Callee – W07 #8

Answer on next slide

```
int function1(int arg) {
  int x, y, w, z, k ;
  x = 15;
  y = 12;
  function2(x);
  z = x + y;
  k = z;
  while (y > 0) { // iterates 12 times
    function2(y-1);
    w = k – 2;
    y--;
  }
  return 0;
}
```

```
int function2(int a) {
   int b, c, d;
   b = a + 1;
   c = a / 2;
   d = b * c;
   printf("%d %d\n", a, c);
   d = d * d;
   return d;
}
```

The architecture has 3 caller, 3 callee registers

function1 : w and z are stored in callee-saved registers, while x, y and k are in caller-saved registers.

function2 : c and d are in callee-saved registers, while b is in a caller-saved register.

- Given the register mapping above, how many ldr/str must be included in the assembly code of these functions to handle the store and restore of registers?
- How many ldr/str must be executed when running this program?

Caller/Callee – W07 #8

```
int function1(int arg) {
  int x, y, w, z, k ;
  x = 15;
  y = 12;
  function2(x);
  z = x + y;
  k = z;
  while (y > 0) { // iterates 12 times
    function2(y-1);
    w = k – 2;
    y--;
  }
  return 0;
}
```

str w,z

str x.y

ldr x.y

str k.y

ldr k.y

ldr w,z

```
int function2(int a) {
   int b, c, d;
   b = a + 1;
   c = a / 2;
   d = b * c;
   printf("%d %d\n", a, c);
   d = d * d;
   return d;
}
```

str c,d

ldr c,d

Additions to the assembly code: 8 ldr/str pairs

Executed ldr/str pairs: 1 (w) + 1(z) + 1(x) + 13 (y) + 12 (k) + 13 (c) + 13 (d)

Floating point - F07Q7

AMD's new SSE5 extension to the x86 ISA includes a new 16 bits floating point datatype. The 16 bits of a number stored in this format are allocated as follows:

- 1 bit sign
- 5 bit exponent with a bias of 15
- 10 bit mantissa (a.k.a. significand)

All other aspects of this format are exactly the same as the standard IEEE floating point studied in class.

- a) Circle the numbers among those listed below that can be represented exactly using the SSE5 floating point format:

  2^{18} too big π irrational 2^{-6} -2^{12} -2^{-12} 1/3 ∞ decimal digits

- b) What decimal number is represented by the following?

  1 (sign) 10111 (exponent) 1001001010 (mantissa) $1.1001001010 \times 2^{8} = 110010010.10$ **Ans: - 402.5**

Conditional ARM assembly - F13Q18

Select the C code that computes the same function as the ARM assembly below

```
LOOP   cmp    r0, r1
       moveq  r2, r3
       addne  r2, r2, #1
       cmp    r2, #10
       blt    LOOP
```

A

```
do {
  if (r0 != r1){
      r2 = r3;
  } else {
      r2++;
  }
}while(r2 >= 10);
```

B

```
do {
  if (r0 == r1){
      r2++;
  } else {
      r2 = r3;
  }
}while(r2 < 10);
```

C

```
do {
  if (r0 == r1){
      r2 = r3;
  } else {
      r2++;
  }
}while(r2 < 10);
```

D

```
while (r2 < 10) {
  if (r0 == r1) {
      r2 = r3;
  } else {
      r2++;
  }
}
```

E

```
while (r2 < 10) {
  if (r0 == r1) {
      r2++;
  } else {
      r2 = r3;
  }
}
```

Linker - Q23F13

Consider the following C program file `fun.c`:

```
        int *a;
        int h = 0;
        struct {
             int x;
             double y;
        }S;
        int foo(int z) {
1.           int *i = a;
2.           h++;
3.           z = *i + 1;
4.           z = bar(10);
5.           return h+z;
        }
```

1. What is the complete set of symbols present in the symbol table of fun.o?
   a, h, S, foo, bar

2. What is the complete set of instruction's locations present in the relocation table of fun.o?
   1, 2, 4, 5

Caller/callee – F11Q9

You are assigned the task of compiling the function below:

```
void mystery_func() {
  int a = 3, b = 4, c = 5, i = 6;
  if (b > 100) {
    return i;
  }

  for (i = 0; i < 10; i++) {
    c = a + i;
    c = another_func(a, b * i);
    printf("Iteration %d: %d %d %d\n", i, a, b, c);
    a = i * i;
  }

  return a;
}
```

Assume that `another_func` is a user-defined function located elsewhere.

Caller/callee – F11Q9 – cont.

1. When we say that register values are stored in/restored from memory by following either the caller-save or the callee-save convention, where in memory do these values usually reside in?

   STACK

2. Next, identify how many store and load operations are **executed** for each variable when mystery_func() is executed **once**, depending on whether the variable is mapped to a caller-saved vs. a callee-saved register.

	Caller-saved		Callee-saved	
	# stores	# loads	# stores	# loads
a	10	10	1	1
b	10	10	1	1
c	0	0	1	1
i	20	20	1	1

Caller/callee – F11Q9 – cont.

3. Finally, assuming that we have 1 caller-saved and 3 callee-saved registers to work on, determine the best assignment (to minimize the total number of load/store operations) for each variable.

Ans: a,b,i: callee-saved

c: caller-saved