EE234

Microprocessor Systems

Final Exam

Dec. 15, 2021. (1:10pm – 4pm)

Instructor: Dae Hyun Kim (daehyun@eecs.wsu.edu)

Name:

WSU ID:

Problem	Points	
1	20	
2	20	
3	20	
4	30	
5	30	
6	20	
Total	140	

Problem #1 (1-D Array, 20 points)

All the registers are 32-bit registers. “int” is a 32-bit signed integer data type. Write an assembly code for the following C code and the given variables.

Variables (both x and y are static arrays.)

- int x[5];
- int y[7];
- &(x[0]) = SP + 4
- &(y[0]) = SP + 40

C code

```
for ( int k = 0 ; k < 5 ; k++ )
  y[k+2] = x[k];
```

(You can use any of R0~R12 for the variable k.)

Problem #2 (1-D Array, 20 points)

All the registers are 32-bit registers. “int” is a 32-bit signed integer data type. Write an assembly code for the following C code and the given variables.

Variables (x is a static array and y is a dynamic array.)

- int x[5];
- int* y = new int[7];
- &(x[0]) = SP + 4
- &y = SP + 40

C code

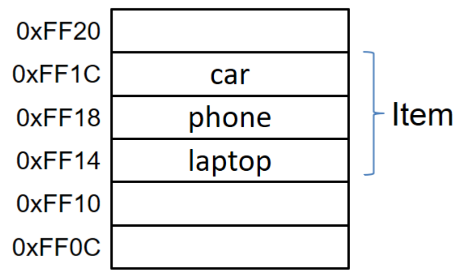
```
for ( int k = 0 ; k < 5 ; k++ )
  y[k+2] = x[k];
```

(You can use any of R0~R12 for the variable k.)

Problem #3 (1-D Array, 20 points)

All the registers are 32-bit registers. “int” is a 32-bit signed integer data type. The following shows a structure definition and how a C/C++ compiler stores the member variables of a structure variable of the data type “Item”. (Notice that the physical addresses shown in the figure don’t matter. I am just showing the relative locations of the member variables in the figure.)

```
struct Item {
  int  laptop;
  int  phone;
  int  car;
};
```



We declare a static array “x” of 10 Item variables as follows:

Item x[10];

Answer the questions below using the following information:

- &(x[3].phone) = 0x0460

(1) What is the address of x[5].car? (5 points)

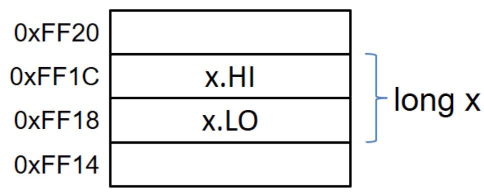
(2) What is the value of &(x[6].laptop)? (5 points)

(3) Is it possible to find the value of x[0].phone from the given information? If yes, what is the value of x[0].phone? If not, just say “not possible”. (5 points)

(4) Is it possible to find the value of the stack pointer register (SP) from the given information? If yes, what is the value of SP? If no, just say “not possible”. (5 points)

Problem #4 (Pointer, 30 points)

All the registers are 32-bit registers. “unsigned int” is a 32-bit unsigned integer data type and “unsigned long” is a 64-bit unsigned integer data type. The following shows how an “unsigned long” variable x is stored in the main memory. The “LO” is the lower 32 bits and the “HI” is the upper 32 bits. The following figure shows how the LO and HI parts of an unsigned long variable are stored in the main memory. (Notice that the physical addresses shown in the figure don’t matter. I am just showing the relative locations of the LO and HI parts.)



Answer the questions below using the following information and the given memory map:

- unsigned int x;
- unsigned long* y;
- SP: 0x7FF0
- &x: SP + 0x0010
- &y: SP + 0x0018

(1) What is the value of x?

(2) What is the value of y?

(3) What is the address of x?

(4) What is the address of y?

(5) What is the value of *((unsigned int*) x)?

(6) What is the value of *y?

(7) What is the value of *((unsigned int*) y)?

unsigned int* k = (unsigned int*) x;

(8) What is the value of k[0]?

(9) What is the value of k[4]?

Address	Data
0x8008	0x4020
0x8004	0x4010
0x8000	0x4000
...	
0x402C	0x8000
0x4028	0x402C
0x4024	0x4028
0x4020	0x4024
0x401C	0x4020
0x4018	0x401C
0x4014	0x4018
0x4010	0x4014
0x400C	0x4010
0x4008	0x400C
0x4004	0x4008
0x4000	0x4004

(10) What is the value of k+5?

```
unsigned long* p = (unsigned long*) x;
```

(11) What is the value of p?

(12) What is the value of p+2?

(13) What is the value of p[3]?

```
x = x + 16;
unsigned int* w = (unsigned int*) x;
```

(14) What is the value of *w?

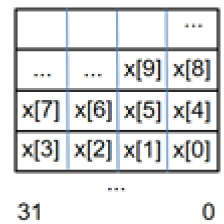
(15) What is the value of w[1]?

Problem #5 (1-D Array, 30 points)

All the registers are 32-bit registers. “unsigned long” is a 64-bit unsigned integer data type and “unsigned char” is an 8-bit unsigned character data type. Write an assembly code for the following C code and the given variables.

Variables (both x and y are static arrays.)

- unsigned char x[8];
- unsigned long y[8];
- &(x[0]) = SP + 8
- &(y[0]) = SP + 80
- The memory map shows how “unsigned char” variables are stored in the main memory.



C code

```
for ( int k = 0 ; k < 8 ; k++ )
  y[k] = x[k];
```

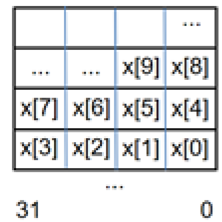
(You can use any of R0~R12 for the variable k. You don’t need to optimize the code.)

Problem #6 (Pointer, 20 points)

All the registers are 32-bit registers. “bool” is a 1-bit data type storing either 0 or 1. Write an assembly code for the following C code and the given variables.

Variables (both x and y are static arrays.)

- bool x[8];
- unsigned int y[8];
- &(x[0]) = SP + 8
- &(y[0]) = SP + 80
- The memory map shows how a “bool” array is stored in the main memory. x[i] is either 0x00 or 0x01.



C code

```
for ( int k = 0 ; k < 8 ; k++ ) {
  if ( y[k] > 10 )
    x[k] = 0;
  else
    x[k] = 1;
}
```

(You can use any of R0~R12 for the variable k. You don’t need to optimize the code.)

Assembly Instructions

R# is a register. (# = 0 ~ 12)

<table>
<tr><th>Instruction</th><th>Meaning</th></tr>
<tr><td>MVN Rd, Ra</td><td>Bitwise inversion (Rd = NOT Ra).
<table>
<tr><td>Before</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>After</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>0</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>AND Rd, Ra, Rb
AND Rd, Ra, #imm
AND Rd, #imm</td><td>Bitwise AND. (Rd = Ra AND Rb), (Rd = Ra AND #imm), (Rd = Rd AND #imm)
<table>
<tr><td>Ra</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td>Rb</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Rd</td><td>0</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>OR Rd, Ra, Rb
OR Rd, Ra, #imm
OR Rd, #imm</td><td>Bitwise OR. (Rd = Ra OR Rb), (Rd = Ra OR #imm), (Rd = Rd OR #imm).
<table>
<tr><td>Ra</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>0</td></tr>
<tr><td>Rb</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Rd</td><td>1</td><td>1</td><td>0</td><td>1</td><td>1</td><td>1</td><td>1</td><td>0</td></tr>
</table></td></tr>
<tr><td>EOR Rd, Ra, Rb
EOR Rd, Ra, #imm
EOR Rd, #imm</td><td>Bitwise exclusive-OR. (Rd = Ra ⊕ Rb), (Rd = Ra ⊕ #imm), (Rd = Rd ⊕ #imm)
<table>
<tr><td>Ra</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td><td>0</td><td>1</td></tr>
<tr><td>Rb</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>1</td><td>0</td></tr>
<tr><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td><td></td></tr>
<tr><td>Rd</td><td>1</td><td>0</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>1</td></tr>
</table></td></tr>
<tr><td>MOV Rd, Ra
MOV Rd, #imm
MOV Rd, Ra, LSR #imm
MOV Rd, Ra, LSR Rx
MOV Rd, Ra, LSL #imm
MOV Rd, Ra, LSL Rx</td><td>Rd = Ra
Rd = #imm
Rd = (Ra >> #imm)
Rd = (Ra >> Rx) where Rx has the # bits to shift Ra to the right.
Rd = (Ra << #imm)
Rd = (Ra << Rx) where Rx has the # bits to shift Ra to the left.</td></tr>
<tr><td>ADD Rd, Ra, Rb
ADD Rd, Ra, #imm
ADD Rd, #imm</td><td>Rd = Ra + Rb
Rd = Ra + #imm
Rd = Rd + #imm</td></tr>
<tr><td>SUB Rd, Ra, Rb
SUB Rd, Ra, #imm
SUB Rd, #imm</td><td>Rd = Ra - Rb
Rd = Ra - #imm
Rd = Rd - #imm</td></tr>
<tr><td>MUL Rd, Ra, Rb
MUL Rd, Ra, #imm</td><td>Rd = Ra * Rb
Rd = Ra * (#imm)</td></tr>
<tr><td>CMP Rd, #imm
CMP Rd, Ra</td><td>Set Z = 1 if Rd == #imm. Otherwise, Z = 0. (Z is the Zero field of the CPSR.)
Set Z = 1 if Rd == Ra. Otherwise, Z = 0.
Notice that N != V is Rd < #imm or Rd < Ra.</td></tr>
<tr><td>BEQ, BNE, BLT, BGE, BGT</td><td>Branch</td></tr>
<tr><td>LDR Rd, [Ra, #imm]</td><td>Load the data stored at [Ra + #imm] to Rd.</td></tr>
<tr><td>STR Rd, [Ra, #imm]</td><td>Store the data stored in Rd to [Ra + #imm].</td></tr>
</table>