EE234

Microprocessor Systems

Midterm Exam 2

Nov. 15, 2023. (2:10pm – 3pm)

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

Name:

WSU ID:

Problem	Points	
1	20	
2	40	
3	40	
Total	100	

Problem #1 (Stack and Subroutines, 20 points)

Correct: +2.5 points. Wrong: -2.5 points. Min: 0.

Assume

- The system we are talking about is single-threaded, single-application.
- Function arguments and return values are processed in the stack.

Answer the following questions.

- The maximum size of the stack in a main memory is dependent on the capacity of the main memory. (True / False)

- The maximum size of the stack in a main memory is dependent on the size of the application code being executed. (True / False)

- If a code contains a recursive function call, it might cause a stack overflow error while the recursive function call is being executed. (True / False)

- If a function has many many many arguments, calling the function might cause a stack overflow error. (True / False)

- If a function has one argument, calling the function will never cause a stack overflow error. (True / False)

- If a function has no argument, calling the function will never cause a stack overflow error. (True / False)

- If a function has no argument and no return value, calling the function will never cause a stack overflow error. (True / False)

- If a function has no argument, no return value, and no function call in it, calling the function will never cause a stack overflow error. (True / False)

Problem #2 (Subroutines and Stack, 40 points)

Answer the following questions for the assembly code shown below.

```
main:
  PUSH {R0}
  SUB SP, SP, #8
  MOV R0, #5
  STR R0, [SP]
  BL com
  LDR R0, [SP, #4]
  ADD SP, SP, #8
  STR R0, [SP, #4]
  POP {R0}
end:
```

```
com:
  PUSH {LR}
  LDR R0, [SP, #4]
  CMP R0, #1
  BGT com_1
  MOV R0, #1
  STR R0, [SP, #8]
  POP {LR}
  BX LR
com_1:
  PUSH {R1,R2,R3,R4}
  MOV R1, #0
  MOV R2, #1
  SUB R3, R0, #1
com_1_loop:
  CMP R2, R3
  BGT com_1_loop_end
  SUB SP, SP, #8
  STR R2, [SP]
  BL com
  LDR R4, [SP, #4]
  ADD SP, SP, #8
  ADD R1, R1, R4
  ADD R2, R2, #1
  B com_1_loop
com_1_loop_end:
  STR R1, [SP, #24]
  POP {R1,R2,R3,R4}
  POP {LR}
  BX LR
```

The main function looks like this in C/C++:

```
int main () {
  int s;
  ..
  s = com(5);
  …
}
```

(1) (30 points) Translate the assembly code of the com() function into a C code. It has one argument and one return value.

(2) (10 points) What is the value stored at [SP+4] when the program ends? (You get 10 points only when you solve the first part above and answer this question correctly.)

Problem #3 (Subroutines and Stack, 40 points)

You should use the following instructions only.

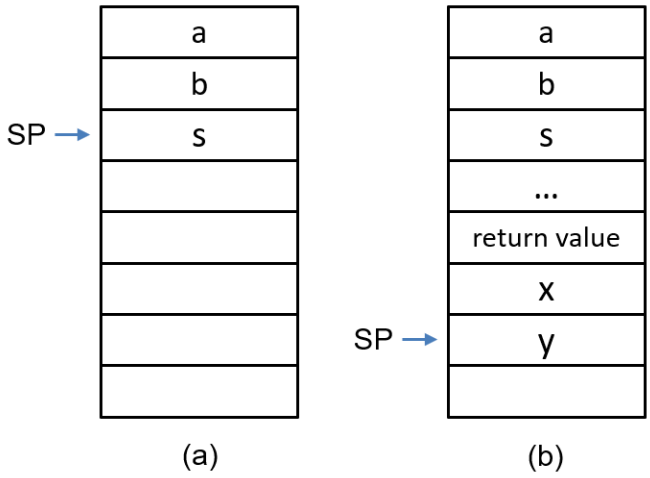
- Instructions
  - ADD, SUB
  - AND, ORR, EOR
  - CMP, BGE/BLT/BGT/BLE/BEQ/BNE
  - B, BL, BX
  - MOV
  - LDR, STR

Write an assembly code for the following C code (the line s=com(a,b) in the main function and the com() function).

```
int main () {
  int s, a, b;
  …
  s = com (a, b);
  …
}
```

```
int com (int x, int y) {
  if ( x < y )
    return x;
  else {
    return com(com(x-y, x+y), com(x-2*y, x+y));
  }
}
```

The following shows the memory map for the function call. (a) in the main function. (b) for the function call.



- In the main function, assume that R0-R12 are being used by other variables (right before the function call s=com(a,b)). This means that you should preserve their values if you want to use any of them.
- Use the stack memory for the function arguments and the return value (shown in (b)).

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 = Bitwise-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</td><td>Bitwise AND. (Rd = Ra AND Rb), (Rd = Ra 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>ORR Rd, Ra, Rb
ORR Rd, Ra, #imm</td><td>Bitwise OR. (Rd = Ra OR Rb), (Rd = Ra 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</td><td>Bitwise exclusive-OR. (Rd = Ra ⊕ Rb), (Rd = Ra ⊕ #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, LSR #imm</td><td>Logical shift right by (#imm) bits. (Rd = Ra >> #imm)
Ex) #imm = 3
<table><tr><td>Before</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>After</td><td>0</td><td>0</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td></tr></table></td></tr>
<tr><td>MOV Rd, Ra, LSL #imm</td><td>Logical shift left by (#imm) bits. (Rd = Ra << #imm)
Ex) #imm = 3
<table><tr><td>Before</td><td>1</td><td>0</td><td>0</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td></tr><tr><td>After</td><td>0</td><td>1</td><td>1</td><td>0</td><td>1</td><td>0</td><td>0</td><td>0</td></tr></table></td></tr>
<tr><td>MOV Rd, Ra
MOV Rd, #imm</td><td>(Rd = Ra)
(Rd = #imm)</td></tr>
<tr><td>ADD Rd, Ra, Rb
ADD Rd, Ra, #imm</td><td>(Rd = Ra + Rb)
(Rd = Ra + #imm)</td></tr>
<tr><td>SUB Rd, Ra, Rb
SUB 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>B [addr]</td><td>Jump to [addr] unconditionally</td></tr>
<tr><td>BEQ, BNE, BLT, BGT, BGE, BLE [addr]</td><td>Branch to [addr] if (BEQ: R1 == R2), (BNE: R1 != R2), (BLT: R1 < R2), (BGT: R1 > R2), (BGE: R1 >= R2), (BLE: R1 <= R2)</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>