Question 1: (8 points)
Using 8-bit two’s complement numbers add −5 with −2
Answer:

```
        8 4 2 1
(2) 0000 0010
    1111 1101  ← 1's
+           1  ← 2's
--------------
   ¹1111 1110
```

(−2) 1111 1110
+(−5) 1111 1011

(−7) 1 1111 1001 : discard carry-out

Question 2: (4 points)
Indicates if the the statements are true or false (T/F)
Answer:

a) The range of negative number (i.e., all possible negative numbers) is larger when using sign magnitude method for representing signed 32-bit numbers. (F)

b) The range of negative number (i.e., all possible negative numbers) is larger when using 2’s complement method for representing signed 32-bit numbers. (T)

c) In both 2’s complement and sign magnitude methods all negative numbers start with 1. (T)

Question 3: (4 points)
Mark the multiplications that can be performed correctly using logical left shift? Assume Z=00001111 in binary.
Answer:

00011110 ⇒ This is mult by 2

Shift left = mult by 2
Shift right = divide by 2

a) Z x 9
b) **Z x 16**

c) Z x 32
d) Z x 64

Question 4: (4 points)
How much should you increment the PC after fetch stage in a processor with byte addressable main memory and width of each instruction is 16 bits?

Answer:

a) 1
b) 2
c) 8
d) 4

2 bytes is 16 bits

Tues Feb 13th slide 6

Question 5: (8 points)
Assume integer variables t1=5 and t2=2. Convert the following C code to MIPS assembly instructions without using multiplication instruction (i.e., mult). Register $t1 and $t2 already contain variable t1 and t2 respectively. All registers and variables to be 32 bits.

t2=t1+4*t2

sll = shift left

$2^n \Rightarrow 2^2 = *4$

Answer:

```
sll $t2, $t2, 2
add $t2, $t2, $t1
```

Question 6 (8 points)

Write down the content of memory and registers (in the box marked with "?") in decimal after all lines of the following assembly code executes. All numbers in the code are in decimal. All registers and variables to be 32 bits.

CPU

r0	Zero
r1	? 120
r2	? 60
r3	? 500

Memory

Address	Data (8 bit)
…	…
60	500
50	54
40	50
30	? 120
…	…
0	…

```
addi $r1,$r0, 120
srl  $r2,$r1, 1
srl  $r3,$r2, 1

sw    $r1, 0($r3)
lw    $r3, 0($r2)
```

does ÷

0

0+120 → r1

r1 = 120

r1/2 = r2

r2 = 60

r3=30

2

r3 = 30

0(30) = 30

120 → 30

→ 0(60) = 60

60 → r3

Answer:

R1=120 (r0 is 0)

R2=60 (right shift divides)

R3=30

Send R1=120 to Mem(30)

Send Mem (60)=500 to R3

Question 7: (4 points)

Which of the following memory technologies are non-volatile?

Answer:

a) DRAM

[✓ b)] Solid state drive

[✓ c)] Flash memory

These can retain data even when there is no power.

d) SRAM
e) Register
[✓ f)] EEPROM

Question 8: (6 points)

Select the correct C code option that will translate to the assembly code below. The operations (add vs subtract) with associated branch (if vs else) should be consistent between C and assembly code. Assume the variables are stored in different registers as the following: a = \$s3, b = \$s4, c = \$s0, d = \$s1, e = \$s2.

a b
s3 ≠ s4 → sub
s3=s4 → add

Assembly code:

```
          a    b
      bne $s3,$s4,else
      add $s0,$s1,$s2
      j Exit     d    e
else: sub $s0,$s1,$s2
Exit:
```

add → if
sub → else

Answer:

Option (a):
```
if (a != b)
        c = d + e;
else
        c = d - e;
```

Option (b):
```
if (a == b)
        c = d - e;
else
        c = d + e;
```

[✓ Option (c):]
```
if (a == b)
        c = d + e;
else
        c = d - e;
```

Option (d):
```
if (a != b)
        c = d - e;
else
        c = d + e;
```

Question 9: (4 points)

Between NOR and NAND flash, which one is more suitable for storing program codes in embedded system? State two reasons.

Answer:

NOR. Fast random read, byte addressable or byte size read.

Check bits

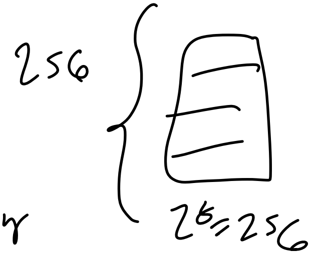
Question 10: (4 pt.) What is the total storage capacity in terms of bits of a memory with 8-bit addresses and addressability of 1 Byte?

Answer:

a) 4096 bits
b) 256 bits
c) 2048 bits
d) 512 bits

256x8

2^n # of unique memory locations for address space



. 1 byte (8 bits) addressability

Question 11: (4 points)
Which of the following is not true for volatile type qualifier?

Answer:

a) The value of volatile variable could be changed by external devices
b) Should be used to represent registers of memory-mapped peripheral
c) Compiler can optimize and remove volatile objects if there is no use in the code
d) Indicates that an object's value is constant

False

Question 12: (4 points)
In memory map of SiFive Fe310 hardware, which memory technology is used for storing Stack, Heap, BSS, data segments during program execution?

Answer:

a) On chip peripherals

b) Off chip nonvolatile memory
c) On chip nonvolatile memory
[✓ d) On chip volatile memory] → SRAM

Question 13: (4 points)
Which of the following memories can be designed with transistors only?
Answer:

a. SRAM (__)
b. DRAM (__)
c. Hard-disk drive (__)
d. Registers (__)

more mechanical devices

Question 14: (4 points)
For the following assembly program, how many times the Addi instruction inside the loop (in blue) will execute? All instructions are based on 32-bit MIPS ISA. Content of $r1 is initially 0. 6 ⇒ executes 3 times

<u>Assembly code:</u>

```
        addi $r1, $r0, 4
        Loop:
        addi $r1, $r1, -2
        bne $r1, $r0, Loop
        sub $r1, $r0, r0
```

→ r0 + 4 → r1 = 4

→ r1 - 2 ⇒ r1 = 2
r1 ≠ r0 _loop → r1 = 2
r1 - 2
= 0
r1 = 0

Answer:

a. 1
b. 2
c. 4
d. 0

hardwired to be zero

Question 15: (6 points)

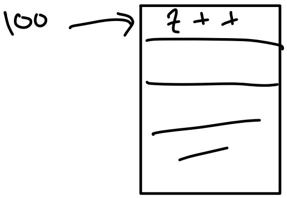
Let us assume that the following code is in the memory and ready for execution. The lines with func() inside main indicate function calls.

If the instruction register (IR) currently contains the first instruction of the func1() procedure, what would is content of return address register at this time?

main → f0 → f1

```
void main(){
 func0();
 z++;
 …
}
```

RA=100

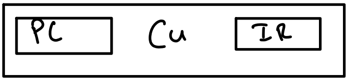


```
int func0(){
 func1();
 x=x+2;
}
int func1(){
 y=y+3;
 …
}
```

RA=2.

RA=add(x=x+2)

→ where you return to



Answer:

a) Memory address of instruction for z++

b) Memory address of instruction for x=x+2

c) Memory address of instruction for y=y+3

d) Memory address of instruction for func1() call inside main()

Question 16: (8 points)
For the C code presented below, indicate the section in memory layout each variable is stored or makes use of. Circle the correct option for each question. (BSS is Uninitialized Data Segment and Data is Initialized Data Segment).

```
int globB=0;
int main () {
  int varA;
  int varB=5;
  static int varC = 1;
  char *varD;
  varD = (char*)malloc(8);
  varA = varC+varB;
  return varA;
}
```

Answer:

a. int globB=0; (Stack—Heap—[✓ *BSS*]—Data)
b. int varA; ([✓ *Stack*]—Heap—BSS—Data)
c. in varB=5; ([✓ *Stack*]—Heap—BSS—Data)
d. static int varC = 1; (Stack—Heap—BSS—[✓ *Data*])
e. char *varD; ([✓ *Stack*]—Heap—BSS—Data)
f. varD = (char*)malloc(8); (Stack—[✓ *Heap*]—BSS—Data)

Question 17: (4 points)
At which stage of the instruction execution cycle, the opcode and operands are identified from the fetched instruction?
Answer:

a) Fetch
[✓ b) Decode]
c) Execute
d) Write back

Question 18: (4 points)

Which are the possible reasons for choosing C language over Java for programming embedded hardware?

a) Improved security against overflow attacks

b) Simpler to write and easier to maintain

c) Low memory requirement

d) Faster code execution

use of pointers for low level memory management

Question 19: (4 points)

What operation is always needed before storing (push operation) something to the stack?

a) Clearing the existing data on the stack

b) Decreasing the stack pointer value → Tues Feb 16th lecture 6

c) Increasing the stack pointer value

d) Storing the return address to the stack pointer register

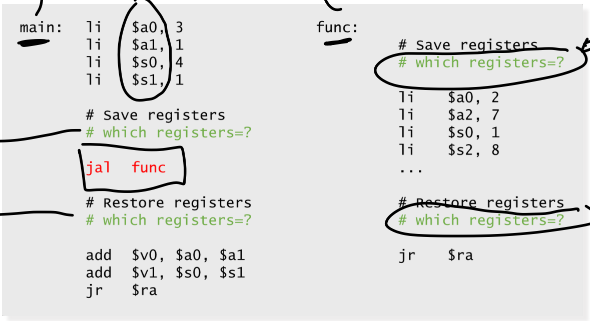
Question 20: (4 points)

For the following example of function call, the commented part of the code with # indicates the segments where certain registers must be stored/restored using the stack.

Write the names of the registers (consider only $a0, $a1, $s0, $s2) under each question mark that must be stored and restored in that region of the code.

caller

callee



```
main:   li   $a0, 3
        li   $a1, 1
        li   $s0, 4
        li   $s1, 1

        # Save registers
        # which registers=?

        jal  func

        # Restore registers
        # which registers=?

        add  $v0, $a0, $a1
        add  $v1, $s0, $s1
        jr   $ra
```

a0, a1

a0, a1

```
func:

    # Save registers
    # which registers=?

    li    $a0, 2
    li    $a2, 7
    li    $s0, 1
    li    $s2, 8
    ...

    # Restore registers
    # which registers=?

    jr    $ra
```

s0, s2

s0, s2