Provider: Li Yuanming

CE/CZ1006 Computer Organization and Architecture

2015-2016 Semester I

Suggested Solution

1.

(a)

R1	SR
0x999	0x004
0x???	0x??? ([0x1FF] is not shown)
0xFFF	0x000
0x72E	0x009
0x063	0x001
0xFC1	0x004

(b)

```
while (Var1 >= Var2) {
    Var1 -= Var2;
}
Var3 = Var1;
```

(c)

```
LOOP  SUB [Var1], [Var2]
      CMP [Var1], [Var2]
      JGE LOOP
      MOV [Var3], [Var1]
```

The problem of the post-test loop is that when Var1 is smaller than Var2 at the initial state, the loop will produce a wrong result.

(d)

Var3 = Var1 % Var2;

2.

(a)

(I1) `PSH [0x200]`

(I2) `PSH [0x201]`

(I3) `PSH 0x202`

(I4) `ADD SP, #3`

(I5) `POPM 6`

(I6) `RET`

(b)

`MOV R1, [SP + 4]`	R1 = VarY
`ADD R1, [SP + 5]`	R1 = VarX + R1
`MOV R2, R1`	R2 = VarX + VarY
`RCN 3`	R2 << 3
`ROL R2`	(R2 × 8)
`ADD R2, R1`	R2 = R2 + R1 ((VarX + VarY) × 9)
`MOV R2, [SP + 3]`	Save result
`MOV [R2], R1`	:

To avoid overflow, we need to discuss both cases: VarX + VarY > 0 and VarX + VarY < 0. We denoted

VarX + VarY as sum.

If sum < 0, 9 × sum ≥ 0x800 => sum ≥ 0xF1D

If sum > 0, 9 × sum ≤ 0x7FF => sum ≤ 0x0E3

Notable, the minimum negative number is 0xFFF!

Therefore, 0x000 ≤ VarX + VarY ≤ 0x0E3 or 0xF1D ≤ VarX + VarY ≤ 0xFFF

(c)

```
SubA  PSH PC                    Save content of R2
      ADD [SP], #4
      JMP SubA
```

First, we push PC into stack. At this time, PC is pointing to ADD [SP], #4. In order to manually shift PC into the right instruction, we add PC by 4 (= length of ADD [SP], #4 + length of JMP SubA). Now PC points to the next instruction of JMP SubA. At this time, we can call JMP SubA.

However, JMP instruction has limitation: the addressing range is from -128 to 127, taking only one instruction word. In contrast, CALL has a range of -2048 ~ 2047, occupying 2 words. Therefore, the distance between the function call and the callee cannot be too far.

3.

(a)

SRAM

Data are stored as long as power is applied. No dynamic refreshing is needed. Each cell consists of 4 ~ 6 transistors, which is larger in size. It is faster in R/W.

DRAM

Data need to be periodically refreshed. Smaller in size, consists of 1 ~ 3 capacitors per cell. Slower in R/W.

(b)

NOR Flash

Supports execution-in-place (XIP), higher cost per bit, faster in read operation.

NAND Flash

Does not support execution-in-place, lower cost per bit, faster in programming operation.

(c)

NOR Flash. System memory needs to support random byte program(write) and read, so that code can run inside the memory. Otherwise, program needs to be transferred to RAM first.

(d)

(i)

Rx message is received by interrupt. Therefore, CPU interrupt latency should be added into per Rx ISR.

Time for execution one ISR =

(CPU Interrupt Latency + instructions in ISR) × instruction cycle time =

(30 + 10) × 10 μs = 400 μs

#ISR = 1 s / 400 μs = 2500

(ii)

Tx message is sent actively. Therefore, only instructions in transmit routine is counted.

10 × 20 μs = 200 μs

#Tx routine = 1 s / 200 μs = 5000

(iii)

2500 < 5000, system A draw back the transfer rate. System B should send no more than 2500 packages / s. In one package, #bits = 1 + 7 + 1 + 1 = 10 bits.

Maximum baud rate = 2500 package/s × 10 bit/package = 25000 bps.

19200 < 25000 < 28800.

Assume each package is sent one after another without any delay, the baud rate should be 19200 bps.

(iv)

In each package, useful data are 7 bits. Therefore, the data transfer rate = 7 × 19200 / 10 = 13400 bps for useful data, or 1680 bytes per second.

As 1680 < 1800, the current system cannot meet the requirement. In this case, we have to decrease the data overhead, namely use a more ‘efficient; UART configuration, for example, 8N1 (8 data bits, no parity scheme, 1 stop bit).

4.

(a)

(i)

Temporal locality and special locality.

Temporal locality

The recently executed code segment will be likely accessed again.

Special locality

Codes/data that are close together are likely used together.

(b)

Page size = 1KiB, #offset bits = 10

The frame address of the physical address 0x0E00 = 0000 1110 0000 0000 -> 0x03

By the mapping in page table, page address = 2.

Total page number = 64, #page bits = 6.

Map physical address to virtual address: 0000 1010 0000 0000 = 0x0A00

(c)

In order to reduce number of calculations, we need to find out which number has more consecutive #0 or #1.

X = 5'd-11 = 5'b1 0101

#consecutive 0 or 1 = 0

Y = 5'd11 = 5'b0 1011

#consecutive 0 or 1 = 1

Therefore, Y is the multiplier and X is the multiplicand.

						1	**0**	**1**	**0**	**1**	X
Carry						**0**	**1**	**0**	**1**	**1**	Y
	0	0	0	0	0	0	1	0	1	1	**-1**
	0	0	0	0	0	0	0	0	0		**0**
	1	1	1	1	0	1	0	1			**+1**
	0	0	0	1	0	1	1				**-1**
	1	1	0	1	0	1					**+1**
1	1	1	1	0	0	0	0	1	1	1	

Thus, the result = **10'b11 1000 0111**

(d)

(i)

I4.

I4 is independent with the rest of the loop body. As long as it is executed after I3, the result will not be affected.

(ii)

I3, I5.

I3 has data dependency with I4. If it is inserted into the branch delay, the result will be wrong.

I5 has data dependency with I6. It should be executed before I6.