Name	Signature	Student Id Nr	*Points*

581305-6 Computer Organization I, miniexam 3, 2.12.2019 (12 p)

Write your answer on this exam paper in the space given. Please notice, that the exam paper is 2-sided.

Follow the recommended subroutine (function) call mechanism. TTK-91 assembly language instructions are:
NOP, STORE, LOAD, IN, OUT, ADD, SUB, MUL, DIV, MOD, AND, IR, XOR, SHL, SHR, COMP, JUMP, JNEG, JZER, JPOS, JNNEG, JNZER, JNPOS, JLES, JEQU, JGRE, JNLES, JNEQU, JNGRE, CALL, EXIT, PUSH, POP, PUSHR, POPR, SVC

Integer-valued 40-element array *T[40]* is defined at main program level. Pointer variable *ptrS* is defined at main program level, and it's value is the address of a 60-element integer array.

Integer valued function *Area(a, b)* computes and returns as its value the area (a*b/2) of right-angled triangle, when the lengths of its legs (catheti) are *a* and *b*. Parameter *a* is call-by-value parameter and *b* is call-by-reference parameter.

Use the symbolic ttk-91 assembly language for answers in part (a)-(d).

a) [2 p] Give a code segment that uses a loop to print the sum of values in array *T*.

```
        load r1, =0    ; sum
        load r2, =0    ; index i
loop    add  r1, T(r2)
        add  r2, =1
        comp r2, =40
        jles  loop
        out r1, =crt
```

b) [2 p] Give a code segment that uses a loop to print the sum of values in the array pointed by *ptrS*.

```
        load r1, =0    ; sum
        load r2, =0    ; index i
        load r3, ptrS  ; pointer to current element in S
loop    add  r1, @r3
        add r3, =1     ; point to next element in S
        add  r2, =1
        comp r2, =60
        jles  loop
        out r1, =crt
```

c) [1 p] Give a code segment that uses function *Area()* to print the area of right-angled triangle with legs 23 and 48.

```
X       dc 48
        .....
        push sp, =0    ; space for return value
        push sp, =23
        push sp, =X    ; address of value 48
        call   sp, Area
        pop  sp, r1    ; get return value
        out r1, =crt
```

-- TURN –

d) [3 p] Implement function *Area(a,b)*.

```
retv    equ -4              ; relative addresses in frame (activation record)
a       equ -3
vb      equ -2
area    pushr  sp           ; save regs
        load r1, a(fp)      ; compute area to r1
        mul  r1, @vb(fp)    ; call-by-reference
        div  r1, =2
        store r1, retv(fp)  ; save area to return value
        popr  sp            ; recover regs
        exit  sp, =2        ; remove space for 2 parameters
```

e) [4 p] Hamming code. The processor reads memory using 64-bit data bus, which is bus is protected with Hamming code.

i. [1 p] How many extra (parity) bits (wires) are needed for Hamming code that *finds* and *fixes* all 1-bit errors and *finds* all 2-bit errors? Explain.
Need parity bits 1, 2, 4, 8, 16, 32, and 64 to find, locate and fix all 1-bit errors,
and extra bit 0 to find all 2-bit errors. So, altogether 8 extra bits (wires) are needed.

ii. [1 p] Who will set the extra bits and when? Who will check their values and when?
Memory unit will set all parity bits before data is placed on the bus. Once the data is received at MMU, the MMU circuits will check whether any errors have occurred. .

iii. [1 p] Assume now, that *one* bit in the data has flipped and become erroneous during data transfer for some reason. How will the system react to this? Is this a big problem? Why?
The MMU circuits will 1st notice that some parity bits are incorrect, then the MMU will add the bit numbers of incorrect parity bits to locate the erroneous bit, flip it to make it correct, and then check whether all parity bits are now ok. They are, because only one bit was wrong. The MMU will cause some interrupt, so that the operating system can keep track of bus errors. One every now and then is expected and not a problem, but more will signify a fault in the bus.

iv. [1 p] Assume now, that *three* bits in the data have flipped and become erroneous during data transfer for some reason. How will the system react to this? Is this a big problem? Why?
The overall parity bit (0) will signify that something is wrong, i.e., odd number of bits have flipped. Errors cannot be located, because parity bits relates to bits in data. With so many errors, you cannot trust, that any other Hamming code parity bits would signal any trouble. It is not a big problem, because we have determined that the probability of three or more random errors happening at the same time is too small to worry about. We have accepted this risk.
(This should probably reported to OS as a likely fault in the bus system. However, if the bus system is faulty, the OS cannot do anything either.)

-- TURN –