

This print-out should have 19 questions. Multiple-choice questions may continue on the next column or page – find all choices before making your selection. The due time is Central time.

EE319K Chapter 8 homework. For information about the C language syntax, see also aLec24.

001 (part 1 of 1) 10 points

In the following C program, what is the effect of the **const** modifier?

```
short AA;
void function(const short BB;){
}

```

1. The **const** modifier changes the situation so the parameter is now only accessible within the scope of the function.
2. The **const** modifier changes the situation so the program runs more efficiently.
3. The **const** modifier changes the situation so the parameter is read-only.
4. None of these choices is correct.
5. The **const** modifier changes the situation so the parameter is now permanently allocated in memory.

002 (part 1 of 1) 10 points

In the following C program, what is the effect of the **const** modifier?

```
const short AA=5;
void function(void){
    short BB;
}

```

1. The **const** modifier changes the situation so the program runs more efficiently.
2. The **const** modifier changes the situation so the variable is read-only.
3. The **const** modifier changes the situation

so the variable is now permanently allocated in memory.

4. The **const** modifier changes the situation so the variable is now only accessible within the scope of this file.

5. None of these choices is correct.

003 (part 1 of 1) 10 points

In the following C program, which variables are local?

```
short AA=4;
const short BB=5;
void function(void){
    short CC;
    for(CC=1;CC<10;CC++){
        short DD;
    }
}

```

1. AA and BB
2. Just AA.
3. None of these choices is correct.
4. Just CC.
5. Just DD.
6. Just BB.
7. CC and DD.
8. BB and CC.

004 (part 1 of 1) 10 points

In the following C program, what is the effect of the **static** modifier?

```
short AA;
void function(void){
    static short BB;
}

```

1. None of these choices is correct.
2. The **static** modifier changes the situation

so the variable is read-only.

3. The **static** modifier changes the situation so the variable is now only accessible within the scope of the function.

4. The **static** modifier changes the situation so the program runs more efficiently.

5. The **static** modifier changes the situation so the variable is now permanently allocated in memory, but the variable is still accessible only within this function.

005 (part 1 of 1) 10 points

In the following C program, what is the effect of the **static** modifier?

```
static short AA;
void function(void){
    short BB;
}
```

1. None of these choices is correct.
2. The **static** modifier changes the situation so the variable is now only accessible within the scope of this file.
3. The **static** modifier changes the situation so the variable is read-only.
4. The **static** modifier changes the situation so the program runs more efficiently.
5. The **static** modifier changes the situation so the variable is now permanently allocated in memory.

006 (part 1 of 1) 10 points

In the following C program, what is the effect of the **static** modifier?

```
short AA;
void static function(void){
    short BB;
}
```

1. The **static** modifier changes the situation so the function is now permanently allocated

in memory.

2. The **static** modifier changes the situation so the function is now only accessible within the scope of this file.

3. The **static** modifier changes the situation so the function is read-only.

4. None of these choices is correct.

5. The **static** modifier changes the situation so the program runs more efficiently.

007 (part 1 of 1) 0 points

Which of the following is **NOT** a rule governing the proper use of the microcomputer stack?

1. Program segments should have an equal number of pushes and pulls.
2. A stack pull should read the memory contents at the address specified by the stack pointer, then increment the stack pointer.
3. All of these rules should be followed.
4. The program should not read or write to memory at addresses less than the stack pointer.
5. The program should not read or write to memory at addresses greater than the stack pointer.
6. A stack push should first decrement the stack pointer, then store the value at the stack pointer address.
7. Stack pushes and pulls should not occur outside the allocated area for the stack.

008 (part 1 of 1) 10 points

What is the difference between **call by value** and **call by reference**?

1. Call by value involves passing the param-

eter in a register and call by reference passes the parameter on the stack.

2. Call by reference involves passing a copy of the number and call by value passes the address of the original data.

3. They are two names for the same thing.

4. None of these choices is correct.

5. Call by value involves passing a copy of the number and call by reference passes the address of the original data.

6. Call by reference involves passing the parameter in a register and call by value passes the parameter on the stack.

009 (part 1 of 1) 10 points

Which assembly code allocates an 8-bit local variable with an initial value of 170?

1. `movb #170, 1, sp`
2. `movb #170, 1, sp+`
3. `movb #170, -1, sp`
4. `movb #170, 1, -sp`
5. `movb #170, 1, +sp`
6. `movb #170, 1, sp-`
7. None of these choices is correct.

010 (part 1 of 1) 10 points

Consider the following assembly subroutine that creates a **local variable** called **buff** of size 23 bytes.

```
buff set  ΔΔ      binding
sub1 pshx        save old frame
     tsx         new frame
     leas -23,s  allocate
     staa buff,x store into
```

```
txs      deallocate
pulx     old frame
rts      return
```

What value should you use in the $\Delta\Delta$ position to implement the proper binding of this local variable?

011 (part 1 of 2) 10 points

Consider the following main program that passes two parameters to a subroutine using the stack.

```
main  lds  #$0c00  initialize stack
loop  ldaa #67
     psha                parameter par1
     ldy  #1070
     pshy                parameter par2
     jsr  sub2          call subroutine
     puly                balance stack
     pula
     bra  loop
par1  set  ∇∇          binding
par2  set  ΔΔ          binding
sub2

     ldab par1,s      read parameter
     ldd  par2,s      read parameter

     rts              return
```

What value should you use in the $\nabla\nabla$ position to implement the proper binding of parameter **par1**?

012 (part 2 of 2) 10 points

What value should you use in the $\Delta\Delta$ position to implement the proper binding of parameter **par2**?

013 (part 1 of 1) 10 points

What is the difference between a **Mealy** finite state machine and a **Moore** finite state machine?

1. None of these choices is correct.

2. The outputs of a **Mealy** finite state machine depend just on the state, while the outputs of a **Moore** finite state machine depend on both the state and the input.

3. The outputs of a **Moore** finite state machine depend just on the state, while the outputs of a **Mealy** finite state machine depend on both the state and the input.

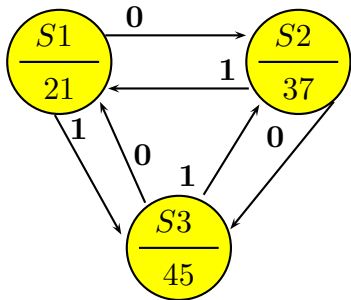
4. A **Moore** finite state machine is more efficient than a **Mealy** finite state machine.

5. The outputs of a **Moore** finite state machine depend just on the state, while the outputs of a **Mealy** finite state machine depend on just the the input.

6. A **Mealy** finite state machine allows for a time delay in each state, while a **Moore** finite state machine does not.

014 (part 1 of 3) 10 points

The following figure shows **Moore** finite state machine. There are three states named **S1 S2 S3**. There is one input signal, connected to PA0, which can be high or low. There is an 8-bit output, connected to PORTB.



The following program is supposed to implement this **Moore** finite state machine.

```

org $F000
S1  nnn 21      Output
    fdb S2,S3   next states
S2  nnn 37      Output
    fdb S3,S1   next states
S3  nnn 45      Output
    fdb S1,S2   next states
FSM movb #$00,DDRA PA0 is input
    movb #$FF,DDRB outputs
    ldx  #S1     State pointer
run  ldab 1,x+   Out value, inc pt
    stab PORTB
    ldab PORTA   Read input
  
```

```

andb #01
lslb          2 bytes/addr
abx          add 0 or 2
ldx  mmm,x   Next
bra  run
org  $FFFE
fdb  FSM     reset vector
  
```

What opcode or pseudo opcode should be placed in the **mmm** position?

1. **rmb.**
2. **movb.**
3. **fdb.**
4. **fcb.**
5. None of these choices is correct.
6. **movw.**

015 (part 2 of 3) 10 points

What value should be placed in the **mmm** position?

1. None of these choices is correct.
2. **1.**
3. **45.**
4. **37.**
5. **21.**
6. **2.**

016 (part 3 of 3) 10 points

After running for a while with a constant input value of 1, what pattern of outputs will occur?

1. **21,37,45,21,37,45,....**
2. **45.**
3. **45,37,21,45,37,21,....**

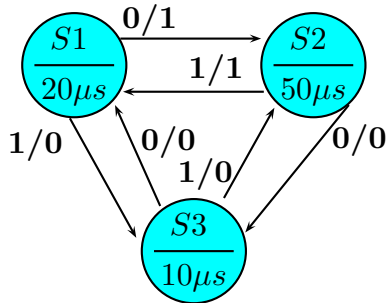
4. 37.

5. None of these choices is correct.

6. 21.

017 (part 1 of 3) 10 points

The following figure shows **Mealy** finite state machine. There are three states named S1 S2 S3. There is one input signal, connected to PJO, which can be high or low. There is a 16-bit time for each state, which contains the time to wait in microseconds. There is a 1-bit output, connected to PH0.



The following program is supposed to implement this **Mealy** finite state machine. Notice the sequence of actions performed by this controller is 1)wait, 2) read input, 3) perform output, and 4) go to next state. You may assume the **Wait** subroutine delays for the number of 125ns cycles as specified in RegD.

```

    org $F000
S1  nnn 160      20us wait
    fcb 1,0      outputs
    fdb S2,S3    next states
S2  nnn 400      50us wait
    fcb 0,1      outputs
    fdb S3,S1    next states
S3  nnn 80       10us wait
    fcb 0,0      outputs
    fdb S1,S2    next states
FSM lds #$0C00   stack
    bclr DDRJ, #PJO PJO is input
    bset DDRH, #PH0 PH0 is output
    ldx #S1      State pointer
run  ldd 2,x+    wait value, inc pt
    jsr Wait
    ldab PORTJ   Read input
  
```

```

andb #P01      B is 0 or 1
ldaa b,x       Out value
staa PORTH     do output
lslb           2 bytes/addr
abx            add 0 or 2
ldx mmm,x      Next
bra run
org $FFFE
fdb FSM        reset vector
  
```

What opcode or pseudo opcode should be placed in the **mmm** position?

1. movw.
2. movb.
3. fcb.
4. fdb.
5. rmb.
6. None of these choices is correct.

018 (part 2 of 3) 10 points

What value should be placed in the **mmm** position?

1. 1.
2. None of these choices is correct.
3. 0.
4. 4.
5. 2.
6. 3.

019 (part 3 of 3) 10 points

After running for a while with a constant input value of 0, what pattern of outputs will occur?

1. A constant low.
2. None of these choices is correct.

3. Can not be determined from this information.

4. A repeating pattern of an output high for 50 us, followed by an output low for 30 us.

5. A constant high.

6. A repeating pattern of an output high for 20 us, followed by an output low for 60 us.