0/44 Questions Answered

Final Exam

STUDENT NAME

Search students by name or email...

Q1 Instructions and Pledge

1 Point

This exam consists of 8 multi-part questions (plus the pledge), and you have 3 hours — budget your time wisely. This is a closed-book, closed-note exam, and "cheat sheets" are not allowed. During the exam you must not refer to the textbook, course materials, notes, or any information on the Internet other than the FAQ and AARCH64 reference linked below. You may not compile or run any code on armlab or any other machine. You may use blank paper as scratch space, but you must enter your answer in the online system in order to receive credit. You are not allowed to communicate with any other person, whether inside or outside the class. You may not send the exam problems to anyone, nor receive them from anyone, nor communicate any information about the problems or their topics.

If you have questions about the wording of some problem, please refer to the FAQ at the following URL:
https://docs.google.com/document/d/e/2PACX-1vQ3tv6HDlQ6HfhJE4NVvnQXDAInCTOcotrOWj_kMJtc3PMXV1ny2K0QbwCnVNw1_1eTxWLwghFALWdg/pub
You may also post a ***private*** *message on Ed, but we only guarantee that we'll be available during the following hours:*

- Friday, 12/17, 12:00 noon – 10:00 PM EST
- Saturday, 12/18, 7:00 PM – 10:00 PM EST
- Sunday, 12/19, 7:00 PM – 10:00 PM EST
- Monday, 12/20, 10:00 AM – 10:00 PM EST

This examination is administered under the Princeton University Honor Code, and by signing the pledge below you promise that you have adhered to the instructions above. ***Please type out the Honor Code pledge exactly as follows, including this exact spelling and punctuation:***

> *I pledge my honor that I have not violated the Honor Code during this examination.*

Enter your answer here

Now type your name as a signature confirming that you have adhered to the Honor Code:

Enter your answer here

Save Answer

Published using Google Docs Learn More Report Abuse

Frequently Asked Questions and Clarifications

The following list will be kept up to date with frequently asked questions and clarifications on the final exam. Please remember to reload this webpage before asking a question!

Questions 4.1 and 4.3: Please go ahead and declare local variables if you need to, despite the fact that they would go after the `asserts`, which is disallowed by C90.

Further clarifications and reminders on 4.x:

- Please respect the instructions that "your functions will not call any other functions".
- Remember to ignore the `asserts` when translating the C code into assembly in questions 4.2 and 4.4.
- There may be ways of solving the problems using instructions that are not in the Assembly Language Reference below. These solutions will receive credit, if they are correct, but using these instructions is definitely not required.

Question 5.4: The C function `pow(x, y)` returns a `double` that is `x` raised to the power `y`.

Question 9: The answer should be a numeric value.

AARCH64 Assembly Language Reference
The following is a reference on AARCH64 assembly language, which you may consult when completing questions 2, 3, 4. ***Please note: this is the only outside reference to which you are allowed to refer during the exam. Attempting to access any other information is a violation of the honor code.***

Registers /Instructions	Description
`x0..x30, xzr / w0..w30, wzr`	8-byte / 4-byte registers. `xzr` and `wzr` hold 0.
`x0..x7 / w0..w7`	Parameters, caller-saved scratch.
`x0 / w0`	Holds return value.
`x19..x28 / w19..w28`	Callee-saved local variables.
`mov dst, src`	Copy `src` (register or immediate value) to dst.
`add/sub/mul dst, src1, src2`	Add / subtract / multiply `src1` by `src2`, storing the result in `dst`.
`adds/subs/muls dst, src1, src2`	Same as above, but also set condition flags.
`sdiv/udiv dst, src1, src2`	Signed / unsigned division.

Published using Google Docs Learn More Report Abuse

`cbz/cbnz src, label`	Branch to `label` if register `src` is zero/nonzero. (Ignores condition flags.)
`b label`	Branch to `label` unconditionally.
`bl label`	Call function at `label` and save return address in `x30`.
`ret`	Return to code at address in `x30`.
`ldr dst, [src]`	Load 8 or 4 bytes (depending on whether `dst` is an `x` or `w` register) into register `dst` from memory at address in register `src`.
`str src, [dst]`	Store to memory at address in register `dst` from register `src`.
`ldrb dst, [src]`	Like `ldr`, but load one byte and zero-extend to size of `dst`.
`strb src, [dst]`	Store one byte.
`[src, offset]`	Register+offset addressing mode. Can be used as the `src` of memory loads (`ldr`, `ldrb`, etc.) or as the `dst` of stores (`str`, `strb`, etc.)
`[src1, src2]`	Register+register addressing mode. Can be used as the `src` of memory loads (`ldr`, `ldrb`, etc.) or as the `dst` of stores (`str`, `strb`, etc.)

Q2 C and Assembly Mix 'n Match

4 Points

For this and the following questions, please refer to the AARCH64 quick reference at the following URL (same as the FAQ):
https://docs.google.com/document/d/e/2PACX-1vQ3tv6HDlQ6HfhJE4NVvnQXDAInCTOcotrOWj_kMJtc3PMXV1ny2K0QbwCnVNw1_1eTxWLwghFALWdg/pub

Please note: this is the only outside reference to which you are allowed to refer during the exam. Attempting to access any other information is a violation of the honor code.

Now study the following four simple functions - two in AARCH64 assembly language and two in C:

```
a:
    cbz     w1, a_1
    udiv    w0, w0, w1
a_1:
    ret
```

```
b:
    ret
```

```
unsigned int c(unsigned int x, unsigned int y)
{
    if (y < x)
        return y;
    else
        return x;
}
```

```
unsigned int d(unsigned int x, unsigned int y)
{
    return x * y;
}
```

For each of the following unknown functions, ***select which of the functions above (a-d) has the same effect***.

Q2.1

1 Point

```
e:
    cmp     w0, w1
    bhs     e_1
    ret
e_1:
    mov     w0, w1
    ret
```

- Same effect as function `a`
- Same effect as function `b`
- Same effect as function `c`
- Same effect as function `d`

Save Answer

Q2.2

1 Point

```
f:
    mov     w2, 0
    cbnz    w1, f_1
    ret
f_1:
    subs    w0, w0, w1
    blo     f_2         // Hint: branches if w0 < w1
    add     w2, w2, 1
    b       f_1
f_2:
    mov     w0, w2
    ret
```

- Same effect as function `a`
- Same effect as function `b`
- Same effect as function `c`
- Same effect as function `d`

Save Answer

Q2.3

1 Point

```
g:
    mov     w2, w0
    mov     w0, wzr
    cbnz    w1, g_1
    ret
g_1:
    add     w0, w0, w2
    sub     w1, w1, 1
    cbnz    w1, g_1
    ret
```

- Same effect as function `a`
- Same effect as function `b`
- Same effect as function `c`
- Same effect as function `d`

Save Answer

Q2.4

1 Point

```
h:
    sub     sp, sp, 16
    str     x30, [sp]
    str     w0, [sp,8]
    str     w1, [sp,12]
    add     w0, w0, w1
    ldr     w1, [sp,12]
    sub     w0, w0, w1
    ldr     x30, [sp]
    add     sp, sp, 16
    ret
```

- Same effect as function `a`
- Same effect as function `b`
- Same effect as function `c`
- Same effect as function `d`

Save Answer

Q3 GCD and LCM

9 Points

In precept, you saw both the C and assembly language code to calculate the GCD (Greatest Common Divisor) of two integers using Euclid's algorithm.

Here is a mildly-edited version of the assembly language code:

```
1       MISSING .SECTION DIRECTIVE
2   promptStr:
3       .string "Enter an integer: "
4   scanfFormatStr:
5       .string "%ld"
6   printfFormatStr:
7       .string "The gcd is %ld\n"

        //--------------------------------------------------------------
        // Return the greatest common divisor of lFirst and lSecond.
        // long gcd(long lFirst, long lSecond)
```

```
        //------------------------------------------------------------
8       .equ    GCD_STACK_BYTECOUNT, 48
9       .equ    LABSSECOND, 8
10      .equ    LABSFIRST, 16
11      .equ    LTEMP,      24
12      .equ    LSECOND,    32
13      .equ    LFIRST,     40

14      .section .text
15      .global gcd

    gcd:
16      sub     sp, sp, GCD_STACK_BYTECOUNT
17      str     x30, [sp]
18      str     x0, [sp, LFIRST]
19      str     x1, [sp, LSECOND]
20      ldr     x0, [sp, LFIRST]
21      bl      labs
22      str     x0, [sp, LABSFIRST]
23      ldr     x0, [sp, LSECOND]
24      bl      labs
25      str     x0, [sp, LABSSECOND]

    gcdLoop:
26      ldr     x0, [sp, LABSSECOND]
27      cmp     x0, 0
28      beq     gcdLoopEnd
29      ldr     x0, [sp, LABSFIRST]
30      ldr     x1, [sp, LABSSECOND]
31      sdiv    x2, x0, x1
32      mul     x3, x2, x1
33      sub     x4, x0, x3
34      str     x4, [sp, LTEMP]
35      ldr     x0, [sp, LABSSECOND]
36      str     x0, [sp, LABSFIRST]
37      ldr     x0, [sp, LTEMP]
38      str     x0, [sp, LABSSECOND]
39      b       gcdLoop

    gcdLoopEnd:
40      ldr     x0, [sp, LABSFIRST]
41      ldr     x30, [sp]
42      add     sp, sp, GCD_STACK_BYTECOUNT
43      ret
```

Q3.1
1 Point

At line 1 of this code, there is a missing `.section` directive. What should it be? ***If multiple options are valid, select the one that best corresponds to idiomatic C code like we've modeled in precept exercises.***

- [ ] `.section .bss`
- [ ] `.section .data`
- [ ] `.section .rodata`
- [ ] `.section .stack`
- [ ] `.section .text`

Save Answer

Q3.2

1 Point

Turning to the `gcd` function itself, it appears that most of the comments are missing. ***Referring to the line numbers above, where would you insert each of the following comments?*** Note that a comment should describe the block of code that ***follows*** it, which should include all loads of variables, computation, stores of results, etc.

`// Prolog`

- [ ] Before line 1
- [ ] Before line 16
- [ ] Before line 17
- [ ] Before line 18
- [ ] Before line 20

Save Answer

Q3.3

1 Point

`// lAbsFirst = labs(lFirst)`

- [ ] Before line 18
- [ ] Before line 20
- [ ] Before line 21
- [ ] Before line 22
- [ ] Before line 29

Save Answer

Q3.4

1 Point

`// lTemp = lAbsFirst % lAbsSecond`

- Before line 29
- Before line 31
- Before line 34
- Before line 35
- Before line 37

Save Answer

Q3.5

1 Point

`// lAbsFirst = lAbsSecond`

- Before line 29
- Before line 30
- Before line 35
- Before line 36
- Before line 38

Save Answer

Q3.6

1 Point

`// Epilog and return lAbsFirst`

- Before line 36
- Before line 40
- Before line 41
- Before line 42
- Before line 43

Save Answer

Q3.7

1 Point

We now want to use the `gcd` code in the computation of the LCM (Least Common Multiple), using the formula
$lcm(i, j) = i * j / gcd(i, j)$.
Here is AARCH64 assembly language code to do the calculation:

```
        //-------------------------------------------------------------
```

```
        // Return the lowest common multiple of lFirst and lSecond.
        // long lcm(long lFirst, long lSecond)
        //--------------------------------------------------------------
1       .equ    LCM_STACK_BYTECOUNT, 48
2       LLCM    .req x23
3       LGCD    .req x22
4       LPROD   .req x21
5       LSECOND .req x20
6       LFIRST  .req x19

7       .section .text
8       .global lcm
    lcm:
        // Prolog
9       sub     sp, sp, LCM_STACK_BYTECOUNT
10      str     x30, [sp]
11      str     x19, [sp, 8]
12      str     x20, [sp, 16]
13      str     x21, [sp, 24]
14      str     x22, [sp, 32]
15      str     x23, [sp, 40]

        // Store parameters in registers
16      mov     LFIRST, x0
17      mov     LSECOND, x1

        // MISSING COMMENT #1
18      bl      gcd
19      mov     LGCD, x0

        // lProd = lFirst * lSecond;
20      MISSING INSTRUCTION

        // MISSING COMMENT #2
21      sdiv    LLCM, LPROD, LGCD

        // Epilog and return lAbsFirst
22      mov     x0, LLCM
23      ldr     x30, [sp]
24      ldr     x19, [sp, 8]
25      ldr     x20, [sp, 16]
26      ldr     x21, [sp, 24]
27      ldr     x22, [sp, 32]
28      ldr     x23, [sp, 40]
29      add     sp, sp, LCM_STACK_BYTECOUNT
30      ret
```

There is a missing comment (#1) before line 18. What should it read to correspond to the probable flattened C code from which the assembly language was generated?

- `gcd();`
- `gcd(lGcd);`
- `lGcd = gcd();`
- `lGcd = gcd(x0, x1);`
- `lGcd = gcd(lFirst, lSecond);`

Save Answer

Q3.8
1 Point

There is a missing instruction on line 20. What should it read?

- ○ `mul LPROD, LFIRST, LSECOND`
- ○ `mul LFIRST, LSECOND, LPROD`
- ○ `mul LPROD, x0, x1`
- ○ `mul x0, x1, LPROD`
- ○ `mul LPROD, [sp,8], [sp,16]`
- ○ `mul [sp,8], [sp,16], LPROD`

Save Answer

Q3.9
1 Point

There is a missing comment (#2) before line 21. What should it read to correspond to the probable flattened C code from which the assembly language was generated?

- ○ // lLcm = lGcd / lProd;
- ○ // lLcm = lProd / lGcd;
- ○ // lLcm = sdiv();
- ○ // lGcd = lProd / lLcm;
- ○ // lGcd = sdiv();

Save Answer

Q4 I thought we were done with questions about strcpy...
12 Points

Recall that the C standard library's `strcpy` function copies a string to a destination from a source, and returns the address of the destination string:

```
char *strcpy(char *dest, const char *src);
```

In Assignment 2, you wrote two C implementations of `Str_copy`, which was intended to mimic `strcpy`. One implementation accessed elements of the source and destination strings by index, changing the index to iterate through the string, while the other implementation accessed elements of the source and destination strings by pointer dereference, and moved the pointer to iterate through the string.

In this problem, you will do the same thing, except in AARCH64 assembly language. You will write both the index-iterating version, `strcpyi`, and the pointer-iterating version, `strcpyp`. To get started, you will first complete the flattened C code for the two functions. You can then compose your assembly by translating your flattened C code; however, you are not required to comment your assembly code with the flattened C statements. Also, recall that the `assert` validation in the C code does not get translated into assembly.

Your assembly functions should ***not*** use either the stack or callee-saved registers to store local variables or saved parameters, but instead do all their work using the caller-saved scratch registers. And since your functions will not call any other functions, they do not need to save x30, and thus do not need to manage a stack frame at all in a prolog or epilog.

Note: each implementation can be completed in about a dozen lines of code or less. If you are writing considerably more than that, you may be off on the wrong track.

Q4.1

2 Points

```
/* Copy string from src to dest using index iteration. Return dest. */
char *strcpyi(char *dest, const char *src) {
     assert(dest != NULL);
     assert(src != NULL);
```

Enter your answer here

```
}
```

Save Answer

Q4.2

4 Points

Hint: use the register+register addressing mode.

```
    // Copy string from src to dest using index iteration. Return dest.
    // char *strcpyi(char *dest, const char *src)
    .global strcpyi
strcpyi:
```

Enter your answer here

Save Answer

Q4.3
2 Points

```
/* Copy string from src to dest using pointer iteration. Return dest. */
char *strcpyp(char *dest, const char *src) {
     assert(dest != NULL);
     assert(src != NULL);
```

Enter your answer here

```
}
```

Save Answer

Q4.4
4 Points

```
    // Copy string from src to dest using pointer iteration. Return dest.
    // char *strcpyp(char *dest, const char *src)
    .global strcpyp
strcpyp:
```

Enter your answer here

Save Answer

Q5 Disassembly and Bit-Twiddling
10 Points

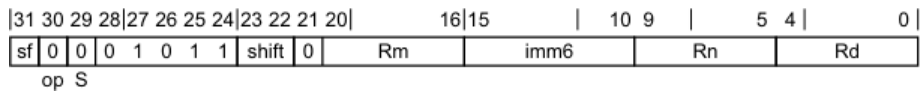
You are writing a *disassembler*: a program that takes AARCH64 machine language instructions and translates them into AARCH64 assembly language. One of the functions you need to write is

```
unsigned int getField(unsigned int uiSrc, unsigned int uiStartBit, unsigned int uiNumBits);
```

to extract each field in the instruction. Its arguments are:

`uiSrc`: a machine-language instruction, represented as a 32-bit unsigned value;
`uiStartBit`: indicates the location in `uiSrc` of the least-significant bit of the field, where `uiStartBit == 0` refers to the least-significant bit of `uiSrc`;
`uiNumBits`: the number of bits in the desired field.

For example, to get the second source register in an `ADD` instruction, which has the format below, we would call `getField(uiSrc, 16, 5)`, since the second source register `Rm` is in a 5-bit field starting at bit 16.



Q5.1

2 Points

What is the value, ***in decimal***, returned by a call to `getField(0x8B130280, 16, 5)`? ***Hint: if you're thinking about endianness, you're overthinking what is necessary to solve this problem.***

Enter your answer here

Save Answer

Q5.2

1 Point

The following are some attempts to implement `getField()`, not all of which are successful. For each one, determine whether it works correctly for all valid inputs, or whether it is buggy. You should consider only valid calls to `getField()` — i.e., you should assume that `uiStartBit + uiNumBits <= 32`. ***Hint: the right-shift operator*** `>>`***, when applied to an*** `unsigned int`***, performs a logical right shift that fills in on the left with "0" bits.***

```
unsigned int getField1(unsigned int uiSrc, unsigned int uiStartBit, unsigned int uiNumBits)
{
    uiSrc << (32 - (uiStartBit + uiNumBits));
    uiSrc >> (32 - (uiStartBit + uiNumBits));
    uiSrc >> uiStartBit;
    return uiSrc;
}
```

- Correct
- Buggy

Save Answer

Q5.3

1 Point

```
unsigned int getField2(unsigned int uiSrc, unsigned int uiStartBit, unsigned int uiNumBits)
{
    uiSrc = uiSrc << (32 - (uiStartBit + uiNumBits));
    uiSrc = uiSrc >> (32 - uiNumBits);
    return uiSrc;
```

```
}
```

- [ ] Correct
- [ ] Buggy

Save Answer

Q5.4

1 Point

```
unsigned int getField3(unsigned int uiSrc, unsigned int uiStartBit, unsigned int uiNumBits)
{
    unsigned int result;
    result = (unsigned int) pow(2, uiNumBits) - 1;
    result &= (uiSrc >> uiStartBit);
    return result;
}
```

- [ ] Correct
- [ ] Buggy

Save Answer

Q5.5

1 Point

```
unsigned int getField4(unsigned int uiSrc, unsigned int uiStartBit, unsigned int uiNumBits)
{
    unsigned int result = 0;
    unsigned int i;

    for (i = 0; i < uiNumBits; i++)
        result = (result << 1) + 1;
    result = result && (uiSrc >> uiStartBit);
    return result;
}
```

- [ ] Correct
- [ ] Buggy

Save Answer

Q5.6

1 Point

```
unsigned int getField5(unsigned int uiSrc, unsigned int uiStartBit, unsigned int uiNumBits)
```

```
{
   unsigned int result = 0;
   result = ~result;
   result = result << (32 - (uiStartBit + uiNumBits));
   result = result >> (32 - (uiStartBit + uiNumBits));
   result = result & uiSrc;
   result = result >> uiStartBit;
   return result;
}
```

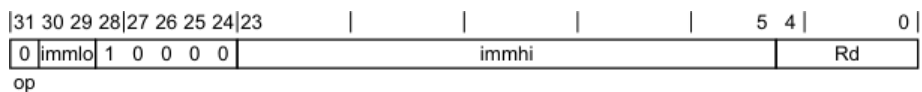
- [ ] Correct
- [ ] Buggy

Save Answer

Q5.7

3 Points

Finally, consider disassembling the `ADR` instruction, which has the format below:



Here is part of the implementation, which relies on a working implementation of `getField()`:

```
void disassem_ADR(unsigned int uiSrc)
{
    unsigned int Rd;
    int offset;

    assert(getField(uiSrc, 31, 1) == 0);
    assert(getField(uiSrc, 24, 5) == 0x10);
    Rd = getField(uiSrc, 0, 5);
    offset = /* INSERT EXPRESSION HERE */;

    /* Now sign-extend offset, and print out the instruction. */
    /* ... */
}
```

Recall that the high-order bits of the `offset` are in the *immhi* field, while its low-order bits are in the *immlo* field. With this in mind, fill in the code marked `/* INSERT EXPRESSION HERE */`. That is, write a single expression for the offset (***before*** it is sign-extended, and ***without*** printing anything), which should involve multiple calls to `getField()` as well as any other necessary arithmetic manipulation:

Enter your answer here

Save Answer

Q6 Linked-List De-Linking

8 Points

Consider the following linked-list type, as well as code intended to remove all nodes containing `val` from the list:

```
struct Node {
    struct Node *next;
    int val;
};
struct List {
    struct Node *first;
};
typedef struct List *List_T;

struct Node *remove_node(struct Node *node, int val)
{
    if (!node) {
        return NULL;
    } else if (node->val == val) {
        struct Node *next = remove_node(node->next, val); /* LINE 0 */
        free(/* EXPRESSION 1 */);
        return /* EXPRESSION 2 */;
    } else {
        node->next = remove_node(node->next, val);
        return /* EXPRESSION 3 */;
    }
}

void List_remove(List_T list, int val)
{
    list->first = remove_node(list->first, val);
}
```

Fill in the missing expressions in the code above. ***Hint: draw a diagram and trace through the execution of one or two simple examples.***

Q6.1

2 Points

What should `EXPRESSION 1` be?

- `node`
- `next`
- `node->next`
- `node->val`
- `NULL`

Save Answer

Q6.2

2 Points

What should `EXPRESSION 2` be?

- ○ `node`
- ○ `next`
- ○ `node->next`
- ○ `node->val`
- ○ `NULL`

Save Answer

Q6.3
2 Points

What should `EXPRESSION 3` be?

- ○ `node`
- ○ `next`
- ○ `node->next`
- ○ `node->val`
- ○ `NULL`

Save Answer

Q6.4
2 Points

We now wish to change the code so that only the ***first*** node containing `val` is removed from the list. How should `LINE 0` be changed (if the remaining code is left unchanged)?

- ○ `struct Node *next = remove_node(node->next, NULL);`
- ○ `struct Node *next = remove_node(NULL, val);`
- ○ `struct Node *next = node;`
- ○ `struct Node *next = node->next;`
- ○ `return node;`

Save Answer

Q7 How smart is the compiler?
5 Points

In lecture, we considered some cases in which a smart optimizing compiler could or could not perform an optimization. Let us consider the function `g` below, and a candidate optimization:

Original function, before optimization:

```
int g(int *x)
{
    return f(x) + f(x);
}
```

After optimization:

```
int g(int *x)
{
    return f(x) << 1;
}
```

Whether or not this optimization is valid depends, in turn, on the function `f`. In each of the following cases, could a smart optimizing compiler perform the optimization above?

Q7.1

1 Point

The following function `f` is defined in the same file as function `g`:

```
int f(int *x)
{
    printf("%d\n", *x);
    return *x + 1;
}
```

- The above optimization to `g` ***is*** allowed
- The above optimization to `g` ***is not*** allowed

Save Answer

Q7.2

1 Point

The following function `f` is defined in the same file as function `g`:

```
int f(int *x)
{
   *x = *x + 1;
   return *x;
}
```

- The above optimization to `g` ***is*** allowed
- The above optimization to `g` ***is not*** allowed

Save Answer

Q7.3

1 Point

The following function `f` is defined in the same file as function `g`:

```
int f(int *x)
{
    return *x + 1;
}
```

- [ ] The above optimization to `g` ***is*** allowed
- [ ] The above optimization to `g` ***is not*** allowed

Save Answer

Q7.4

1 Point

The following function `f` is defined in the same file as function `g`:

```
static int counter = 0;
int f(int *x)
{
    counter++;
    return *x + 1;
}
```

- [ ] The above optimization to `g` ***is*** allowed
- [ ] The above optimization to `g` ***is not*** allowed

Save Answer

Q7.5

1 Point

The function `f` is ***not*** defined in the same file as function `g`.
Its definition is known only at link time.

- [ ] The above optimization to `g` ***is*** allowed
- [ ] The above optimization to `g` ***is not*** allowed

Save Answer

Q8 (Re-)Make Me!

9 Points

Here are fragments of modules that will be built into one executable named `testtable`. All pertinent information is shown.

```
/* testtable.c */
#include <stdio.h>
#include "table.h"
... rest of testtable.c

/* table.h */
#ifndef TABLE_INCLUDED
#define TABLE_INCLUDED
#include <stddef.h>
#include "mydefs.h"
... rest of table.h
#endif

/* table.c */
#include "table.h"
#include "node.h"
... rest of table.c

/* node.h */
#ifndef NODE_INCLUDED
#define NODE_INCLUDED
#include "mydefs.h"
... rest of node.h
#endif

/* node.c */
#include "node.h"
... rest of node.c

/* mydefs.h */
#ifndef MYDEFS_INCLUDED
#define MYDEFS_INCLUDED
... rest of mydefs.h
#endif
```

You have written a `Makefile` for this project that follows COS 217 best practices. Its structure is as follows:

```
TARGET1: DEPENDENCIES1
        gcc217 testtable.o table.o node.o -o testtable
TARGET2: DEPENDENCIES2
        gcc217 -c testtable.c
TARGET3: DEPENDENCIES3
        gcc217 -c table.c
TARGET4: DEPENDENCIES4
        gcc217 -c node.c
```

Answer the following questions about the `Makefile` and the behavior of `make`:

Q8.1
1 Point

The line `TARGET1: DEPENDENCIES1` should be `testtable: testtable.o table.o node.o table.h node.h mydefs.h`

- True
- False

Save Answer

Q8.2
1 Point

The dependency rule for `testtable.o` should be `testtable.o: testtable.c table.h mydefs.h`

- True
- False

Save Answer

Q8.3
1 Point

The dependency rule for `table.o` should be `table.o: table.c table.h stddef.h mydefs.h node.h mydefs.h`

- True
- False

Save Answer

Q8.4
1 Point

The dependency rule for `node.o` should be `node.o: node.c node.h mydefs.h`

- True
- False

Save Answer

Q8.5
1 Point

The command `gcc217 -c testtable.c` builds `testtable`.

- True
- False

Save Answer

Q8.6

1 Point

When `mydefs.h` is out of date, `table.o` gets built twice, because `table.c` depends on `mydefs.h` through both `table.h` and `node.h`.

- True
- False

Save Answer

Q8.7

1 Point

After issuing the `make` command to build the program, we modify the definition of one function in `node.c`. If we issue the `make` command now, how many of the four targets in the `Makefile` will be rebuilt?

- 0
- 1
- 2
- 3
- 4

Save Answer

Q8.8

1 Point

After issuing the `make` command to build the program, we execute the `touch mydefs.h` command to update the timestamp for `mydefs.h` to the current time, without changing its contents. If we issue the `make` command now, how many of the four targets in the `Makefile` will be rebuilt?

- 0
- 1
- 2
- 3
- 4

Save Answer

Q8.9
1 Point

If we were to rearrange the `Makefile` by swapping lines 1 and 2 (TARGET1 and its command) with lines 3 and 4 (TARGET2 and its command), then `make` would behave the same way in all situations as it does with the original `Makefile`.

- True
- False

Save Answer

Q9 Beat the Grader!
3 Points

The following function returns a grade. You will receive 0 points if it returns an `'F'`, 1 point if it returns a `'C'`, 2 points if it returns a `'B'`, and 3 points if it returns an `'A'`. ***Note: no buffer overrun is necessary on this problem. Please do not attempt to execute one.***

```
char grader(unsigned char secret)
{
    const int princeton = -1746;
    const int beat_harvard = 18-16;
    const int beat_yale = 35-20;
    int i;

    if ((unsigned int) princeton < secret)
        return 'F';

    for (i = 0; i < 3; i++) {
        if (secret < beat_harvard || secret < beat_yale)
            return 'C';
        secret -= beat_harvard;
        secret += beat_yale;
    }

    if (secret)
        return 'B';
    else
        return 'A';
}
```

What value would you like to pass as the `secret`?

Enter your answer here

Save Answer