

# NET3001 Fall 09

## Assignment 2

**Due:** Oct 1, beginning of class

**Submitting:** Use the submit.exe program. The files should be named  
assign21.s  
assign22.s  
assign23.s

In Eclipse, make a project for each, called assign21, assign22 and assign23.

### Q1 Modem Control System: subroutine calls & globals

[8 marks]

In this first question, imagine you are part of a design team working on a system controller for a DSL modem with VOIP. Your code must monitor the available bandwidth, monitor the number of VOIP channels that are active, and reduce the allowable data (non-VOIP) bandwidth to guarantee good VOIP operation.

For example, at some point in the day, the available bandwidth may be 640kb/s (kilobits per second), which is about 80kB/s (kiloBytes per second). Each VOIP channel requires 10kB/s (8kB/s for speech, and 2kB/s for overhead).

In addition, you should reserve 2kB/s of bandwidth for controlling the VOIP calls; this 2kB/s reserve is constant, no matter how many VOIP calls are happening, even 0 calls.

As VOIP calls appear, your code must reduce the available bandwidth for data traffic, making room for the voice call(s). For example, if 640kb/s (80kB/s) of total bandwidth is available, when the first VOIP calls happens, you should set the data traffic bandwidth to 68kB/s. As VOIP calls finish, your code should increase the available data bandwidth again.

You must write your code in MSP430 assembly language. Certain RAM location will contain the information that you need to process; here we're using *global* variables to communicate with the subroutine.

Address	Name	Contents
0x200	totalBWbits	[unsigned word: input] the current total available bandwidth that the ADSL modem can handle; this is given in kb/s, and will usually be about 640kb/s; it will vary over time, and may go as low as 200kb/s
0x202	dataBW	[unsigned word: output] you should write the current data (non-VOIP) bandwidth allowance into this word; this is given in kB/s, and will usually be 40-68kB/s, but may go much lower, even to zero
0x210	numCalls	[unsigned byte: input] the current number of VOIP calls taking place

Use the assembly language as described in class. You should compile the code in the Eclipse

environment.

Write your code as a subroutine. Your routine will be called once a second by the master control code in the DSL modem. You can use the code attached below as your `main()`; if you call your subroutine `bwCalculate()`, then you can simply attach the code below. In other words, the assignment code is the master, and your code is a subroutine which the master will call, once per second. Your code should ensure that `dataBW` never goes negative.

Run the simulator and watch the contents of address `0x200`, `0x202` and `0x210` change. Note the maximum and minimum values for `dataBW`.

```
.data
.global totalBWbits, dataBW, numCalls
totalBWbits: .word 640      ; kb/s total bw; ranges from 0..640
dataBW:      .word 0       ; kB/s bw available for data [SET BY YOUR CODE]
.org 0x10
numCalls:    .byte 0       ; number of VOIP calls

.text
.global main
main:
mov #0x600,r1      ; load the stack pointer
mov #0, r4         ; r4 will count up from 0 to 16
mainLoop:         ; top of the "for" loop: for(r4=0; r4<16; r4++)
mov r4,r6
add r6,r6         ; r6 is 2*r4
mov rateTable(r6),&totalBWbits ; drop the data into the 2 ram locations
mov.b callTable(r4), &numCalls

call #bwCalculate ; calculate the available bandwidth

inc r4
cmp #16,r4
jne mainLoop
jmp main          ; start all over again

.set NUM_TEST, 16
rateTable: .word 640,640,640,640, 620,590,560,520, 480,450,380,330
           .word 360,430,500,220
callTable: .byte 0,0,1,2, 2,3,2,2, 2,1,1,0, 0,0,0,4

[insert your code & data here]

.section .bss
.byte 0
.section .vectors, "a"
.org 0x1e
.word main
```

Run the code in the simulator. Don't forget to set the `msp430-gcc` assembler options to include `--gstabs+`. Here's a suitable `.gdbinit`:

```
target sim
load Debug/assign2
```

## Q2 Packet filtering: bit testing, arguments in registers

[8 marks]

In the above environment, write a subroutine called `packetFilter()` to filter the actual packets. This subroutine accepts two parameters:

- a pointer to an ethernet packet (in R15)
- an unsigned word which is the packet length (in R14)

The subroutine returns a 1 or a 0 in R15, to indicate whether the packet should be sent, or delayed.

Here are the rules:

- a) any packet which is not IP should be sent (return a 1 in R15)
- b) any packet which is IP, and which has the Q-bits set, should be sent (return a 1 in R15); these are VOIP packets
- c) if the current number of bytes sent in this 1 second period is less than the data allowance, go ahead and send the data packet (return a 1 in R15)
- d) if the current number of bytes exceeds the data allowance, delay the data packet (return a 0 in R15)

As each packet is sent, increase the number of bytes sent.

Address	Name	Contents
0x200	totalBWbits	[unsigned word: input] the current total available bandwidth that the ADSL modem can handle; this is given in kb/s, and will usually be about 640kb/s; it will vary over time, and may go as low as 200kb/s
0x202	dataBW	[unsigned word: output] you should write the current data bandwidth allowance into this word; this is given in kB/s, and will usually be 40-68kB/s, but may go much lower, even to zero
0x204	sentBytesLo	[unsigned word] the number of bytes sent in the current 1 second; this word holds the low 3 decimal digits of the total
0x206	sentBytesHi	[unsigned word] the number of kilo bytes sent in the current 1 second
0x210	numCalls	[unsigned byte: input] the current number of VOIP calls taking place

Because the number of bytes sent can range from 0...80,000, a single 16 bit word is not enough to hold the information. So instead, use the word at 0x204 to hold the bottom 3 decimal digits; use the word at 0x206 to hold the number of kBytes. To handle this math, every time you have to send  $n$  bytes, simply add  $n$  to the word at 0x204. If the result of that add is  $>1000$ , then subtract 1000 from 0x204, and add 1 to 0x206.

The router implements packet level 2 priority bits (802.1 p/Q). It *always* allows these packets through. All non "tcp/udp" packets are allowed through. All "tcp/udp" packets are allowed through only until the number of bytes sent matches the number stored in address 0x202.

You can identify low-priority tcp/udp packets by examining the data. For high priority VOIP packets, the bytes at offset 12,13 and 14 will be 0x81, 0x00 and 0x80...0xFF. If the bytes at offset 12 and 13 are 0x08, 0x00, this is a plain old IPv4 tcp/udp packet, and is low priority. The low-priority IPv6 packets have bytes 12 and 13 as 0x86, 0xdd.

Any other pattern is a network control packet (such as ARP) and should be sent.

The byte count at 0x204/0x206 needs to be reset every second. To signal this, the program which is calling your subroutine will call it with a 0 in R15; this is not a valid pointer address. Your code should detect a 0 in R15, and reset both 0x204 and 0x206 to 0. Return a 1 in R15.

The test harness given below should be used to test your solution to this assignment. It will call your two subroutines: `bwCalculate()` and `packetFilter()` repeatedly. Every one second, it will call `bwCalculate()`, and then it will call `packetFilter()` with a 0, asking for a reset. In between, it will call `packetFilter()` many times, with various kinds of packets: some ARP, some TCP, some UDP and some UDP-VOIP.

Every time you allow a packet to be sent, add the length of the packet to `sentBytesLo`. If `sentBytesLo` goes over 1000, then subtract 1000 from it, and add one to `sentBytesHi`. This way you can keep track of the number of kBytes of data that is sent. *[YOU MAY CHOSE TO ADD UP ONLY THE NON-VOIP DATA BYTES THAT ARE SENT; FULL MARKS FOR EITHER VERSION.]*

This is a realistic example. The channel described here is a typical DSL stream, in the direction from a residence to the ISP. Of course, the other direction is much faster; this assignment does not deal with the other direction.

For this part of the assignment, the information is passed into your subroutine in r14 and r15. You may change r12, r13, r14 and r15; it's not important to preserve them. Make sure you preserve *all* the registers r4...r11.

```
.data
.global totalBWbits, dataBW, numCalls
totalBWbits: .word 640          ; kb/s total bw; ranges from 0..640
dataBW:      .word 0           ; kB/s bw available for data [SET BY YOUR CODE]
sentBytesLo: .word 0           ; low part of the total byte count this second
sentBytesHi: .word 0           ; high part of total byte count, kBytes
.org 0x10
numCalls:   .byte 0            ; number of VOIP calls

.text
.global main
main:
    mov #0x600,r1              ; load the stack pointer
    mov #0,r4                  ; for (r4=0; r4<12; r4++)

    mov #4, &numCalls          ; 4 active calls
    mov #352, &totalBWbits     ; a poor ADSL connection
    call #bwCalculate
    mov #0, r15
    call #packetFilter

main2_loop:
    mov r4, r6
    add r6, r6 ; r6 = 2*r4
    mov pkt_ptr(r6), r15      ; r15 = address of packet
    mov packet_lens(r6), r14  ; r14 = packet len
```

```

call #packetFilter

inc r4
cmp #12, r4
jnz main2_loop      ; end of for loop
jmp main

[insert your code here... 2 subroutines]
; packet data

pkt_ptr: .word packet_http_req, packet_404, packet_arp, packet_voip_udp
         .word packet_http_req, packet_404, packet_arp, packet_voip_udp
         .word packet_404, packet_404, packet_404, packet_404
packet_lens: .word packet_404-packet_http_req
            .word packet_arp-packet_404
            .word packet_voip_udp-packet_arp
            .word packet_end-packet_voip_udp
            .word packet_404-packet_http_req
            .word packet_arp-packet_404
            .word packet_voip_udp-packet_arp
            .word packet_end-packet_voip_udp
            .word packet_arp-packet_404
            .word packet_arp-packet_404
            .word packet_arp-packet_404
            .word packet_arp-packet_404
packet_http_req:
; ether header
.byte 0x00,0x1d, 0xb3,0x84, 0x3f,0x03, 0x00,0x0e, 0x35,0x4c, 0xf7,0x91, 0x08,0x00
; ip header, len=0x96
.byte 0x45,0x00
.byte 0x00,0x96, 0x58,0xe9, 0x40,0x00, 0x40,0x06, 0x76,0x74
; src/dest ip addr
.byte 0xc0,0xa8, 0xd7,0x89, 0xc0,0x96, 0x12,0x3c
; tcp header: src/dest port
.byte 0xe9,0xd7, 0x00,0x50, 0x88,0x10, 0x9f,0xfb, 0xea,0x27, 0xf6,0x52, 0x50,0x18
.byte 0x16,0xd0, 0x21,0x58, 0x00,0x00
; payload "GET /blah.html....."
.byte 0x47,0x45, 0x54,0x20, 0x2f,0x62, 0x6c,0x61, 0x68,0x2e
.byte 0x68,0x74, 0x6d,0x6c, 0x20,0x48, 0x54,0x54, 0x50,0x2f, 0x31,0x2e, 0x30,0x0d, 0x0a,0x55
.byte 0x73,0x65, 0x72,0x2d, 0x41,0x67, 0x65,0x6e, 0x74,0x3a, 0x20,0x57, 0x67,0x65, 0x74,0x2f
.byte 0x31,0x2e, 0x31,0x31, 0x2e,0x34, 0x0d,0x0a, 0x41,0x63, 0x63,0x65, 0x70,0x74, 0x3a,0x20
.byte 0x2a,0x2f, 0x2a,0x0d, 0x0a,0x48, 0x6f,0x73, 0x74,0x3a, 0x20,0x77, 0x77,0x77, 0x2e,0x61
.byte 0x64,0x6f, 0x62,0x65, 0x2e,0x63, 0x6f,0x6d, 0x0d,0x0a, 0x43,0x6f, 0x6e,0x6e, 0x65,0x63
.byte 0x74,0x69, 0x6f,0x6e, 0x3a,0x20, 0x4b,0x65, 0x65,0x70, 0x2d,0x41, 0x6c,0x69, 0x76,0x65
.byte 0x0d,0x0a, 0x0d,0x0a

packet_404:
; ether header, tcp
.byte 0x00,0x0e, 0x35,0x4c, 0xf7,0x91, 0x00,0x1d, 0xb3,0x84, 0x3f,0x03, 0x08,0x00, 0x45,0x00
.byte 0x01,0x0a, 0x0e,0x1a, 0x40,0x00, 0xef,0x06, 0x11,0xcf, 0xc0,0x96, 0x12,0x3c, 0xc0,0xa8
.byte 0xd7,0x89, 0x00,0x50, 0xe9,0xd7, 0xea,0x27, 0xf6,0x52, 0x88,0x10, 0xa0,0x69, 0x50,0x10
.byte 0x11,0x8a, 0x6b,0x8e, 0x00,0x00
; payload: "HTTP/1.1 404 Not Found...."
.byte 0x48,0x54, 0x54,0x50, 0x2f,0x31, 0x2e,0x31, 0x20,0x34
.byte 0x30,0x34, 0x20,0x4e, 0x6f,0x74, 0x20,0x46, 0x6f,0x75, 0x6e,0x64, 0x0d,0x0a, 0x44,0x61
.byte 0x74,0x65, 0x3a,0x20, 0x57,0x65, 0x64,0x2c, 0x20,0x31, 0x20,0x31, 0x36,0x20, 0x53,0x65, 0x70,0x20
.byte 0x32,0x30, 0x30,0x39, 0x20,0x32, 0x31,0x3a, 0x34,0x30, 0x3a,0x33, 0x37,0x20, 0x47,0x4d
.byte 0x54,0x0d, 0x0a,0x53, 0x65,0x72, 0x76,0x65, 0x72,0x3a, 0x20,0x41, 0x70,0x61, 0x63,0x68
.byte 0x65,0x0d, 0x0a,0x43, 0x6f,0x6e, 0x74,0x65, 0x6e,0x74, 0x2d,0x54, 0x79,0x70, 0x65,0x3a
.byte 0x20,0x74, 0x65,0x78, 0x74,0x2f, 0x68,0x74, 0x6d,0x6c, 0x0d,0x0a, 0x43,0x61, 0x63,0x68
.byte 0x65,0x2d, 0x43,0x6f, 0x6e,0x74, 0x72,0x6f, 0x6c,0x3a, 0x20,0x6d, 0x61,0x78, 0x2d,0x61
.byte 0x67,0x65, 0x3d,0x39, 0x30,0x30, 0x0d,0x0a, 0x45,0x78, 0x70,0x69, 0x72,0x65, 0x73,0x3a
.byte 0x20,0x57, 0x65,0x64, 0x2c,0x20, 0x31,0x36, 0x20,0x53, 0x65,0x70, 0x20,0x32, 0x30,0x30
.byte 0x39,0x20, 0x32,0x31, 0x3a,0x35, 0x35,0x3a, 0x33,0x37, 0x20,0x47, 0x4d,0x54, 0x0d,0x0a
.byte 0x43,0x6f, 0x6e,0x6e, 0x65,0x63, 0x74,0x69, 0x6f,0x6e, 0x3a,0x20, 0x63,0x6c, 0x6f,0x73
.byte 0x65,0x0d, 0x0a,0x56, 0x61,0x72, 0x79,0x3a, 0x20,0x41, 0x63,0x63, 0x65,0x70, 0x74,0x2d
.byte 0x45,0x6e, 0x63,0x6f, 0x64,0x69, 0x6e,0x67, 0x2c,0x20, 0x55,0x73, 0x65,0x72, 0x2d,0x41
.byte 0x67,0x65, 0x6e,0x74, 0x0d,0x0a, 0x0d,0x0a

packet_arp: ; arp who-has 192.168.215.205 tell 192.168.215.20
.byte 0xff,0xff, 0xff,0xff, 0xff,0xff, 0x08,0x00, 0x0f,0x1e, 0x7e,0x6d, 0x08,0x06, 0x00,0x01
.byte 0x08,0x00, 0x06,0x04, 0x00,0x01, 0x08,0x00, 0x0f,0x1e, 0x7e,0x6d, 0xc0,0xa8, 0xd7,0x14
.byte 0x00,0x00, 0x00,0x00, 0x00,0x00, 0xc0,0xa8, 0xd7,0xcd, 0x00,0x00, 0x00,0x00
.byte 0x00,0x00, 0x00,0x00, 0x00,0x00, 0x00,0x00, 0x01,0xb3, 0xee,0x71

packet_voip_udp:
.byte 0x00,0x1d, 0xb3,0x84, 0x3f,0x03, 0x00,0x0e, 0x35,0x4c, 0xf7,0x91, 0x81,0x00, 0xc0,0x00
.byte 0x08,0x00, 0x45,0x00
.byte 0x00,0xc8, 0x00,0x00, 0x40,0x00, 0x40,0x11, 0xe0,0x41, 0xc0,0xa8, 0xd7,0x89, 0xc0,0xa8
.byte 0x01,0x09, 0x1f,0x40, 0x1f,0x4d, 0x00,0xb4, 0x7f,0x5d, 0x80,0x00, 0x82,0x78, 0xb2,0x34
.byte 0x4d,0x69, 0x27,0x29, 0x35,0xf0, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e

```

```

.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e

```

packet\_end:

```

.section .bss
.byte 0
.section .vectors, "a"
.org 0x1e
.word main

```

### Q3: Checksums: loops, parameters on stack

[8 marks]

For the DSL modem, you have been asked to test two kinds of packet checksums. The first kind simply adds up all the bytes in the packet, and returns the sum as a byte; call this one `csumBytes`. The second kind adds up all the *words* (2 byte pairs) in a packet, and returns the sum as a *word*; call this one `csumWords`. Your tasks are:

- write each subroutine
- test it with the test harness code given (you can compare results with your classmates)
- calculate the number of instructions required for each subroutine to process a packet of 1500 bytes

[One particular condition applies, though. When you write `csumWords`, you must process the words in network order, that is in big-Endian. In the msp430, when you read a word from memory, it is assumed to be in little-Endian format. So before you add it to the checksum, you must swap the bytes to make it big-Endian. Then, just before you return from the subroutine, you will have to swap the bytes again. For example, if your packet contains:

01 02 03 04

then you will read the first word as 0x0201. Swap that around to make it 0x0102. Then read the second word; it will be 0x0403. Swap that one as well, to make it 0x0304. Now add them, you should get 0x0406. When you return from subroutine, swap that again to return 0x0604.

You can assume that the length of the packet will always be an even number; you don't need to check.]

For this part of the assignment, the pointer to the packet will be stored on the stack, and the length of the packet will be stored on the stack. By the time execution gets to your subroutine, the stack will look like this:

- pointer to data packet [2 bytes]
- length of the packet [2 bytes]
- sp-> -return address [2 bytes]

You should return the checksum in r15. Preserve all other registers.

Here is your test harness:

```
.text
.global main
main:
    mov #0x600, r1    ; initialize the stack
    mov #0, r4       ; for (r4=0; r4<4; r4++)

main3_loop:
    mov r4, r6
    add r6, r6       ; r6 = 2*r4
    push pkt_ptr(r6) ; address of packet
    push packet_lens(r6) ; packet len

    call #csumBytes

    call #csumWords

    add #4, r1       ; pop 2 things
    inc r4
    cmp #4, r4
    jl main3_loop   ; end of for loop
    jmp main

    [ insert your code here ]

; packet data
pkt_ptr: .word packet_http_req, packet_404, packet_arp, packet_voip_udp
         .word packet_http_req, packet_404, packet_arp, packet_voip_udp
         .word packet_404, packet_404, packet_404, packet_404
packet_lens: .word packet_404-packet_http_req
            .word packet_arp-packet_404
            .word packet_voip_udp-packet_arp
            .word packet_end-packet_voip_udp
            .word packet_404-packet_http_req
            .word packet_arp-packet_404
            .word packet_voip_udp-packet_arp
            .word packet_end-packet_voip_udp
            .word packet_arp-packet_404
            .word packet_arp-packet_404
            .word packet_arp-packet_404
            .word packet_arp-packet_404
packet_http_req:
    ; ether header
    .byte 0x00,0x1d, 0xb3,0x84, 0x3f,0x03, 0x00,0x0e, 0x35,0x4c, 0xf7,0x91, 0x08,0x00
    ; ip header, len=0x96
    .byte 0x45,0x00
    .byte 0x00,0x96, 0x58,0xe9, 0x40,0x00, 0x40,0x06, 0x76,0x74
    ; src/dest ip addr
    .byte 0xc0,0xa8, 0xd7,0x89, 0xc0,0x96, 0x12,0x3c
    ; tcp header: src/dest port
    .byte 0xe9,0xd7, 0x00,0x50, 0x88,0x10, 0x9f,0xfb, 0xea,0x27, 0xf6,0x52, 0x50,0x18
    .byte 0x16,0xd0, 0x21,0x58, 0x00,0x00
    ; payload "GET /blah.html...."
    .byte 0x47,0x45, 0x54,0x20, 0x2f,0x62, 0x6c,0x61, 0x68,0x2e
    .byte 0x68,0x74, 0x6d,0x6c, 0x20,0x48, 0x54,0x54, 0x50,0x2f, 0x31,0x2e, 0x30,0x0d, 0x0a,0x55
    .byte 0x73,0x65, 0x72,0x2d, 0x41,0x67, 0x65,0x6e, 0x74,0x3a, 0x20,0x57, 0x67,0x65, 0x74,0x2f
    .byte 0x31,0x2e, 0x31,0x31, 0x2e,0x34, 0x0d,0x0a, 0x41,0x63, 0x63,0x65, 0x70,0x74, 0x3a,0x20
    .byte 0x2a,0x2f, 0x2a,0x0d, 0x0a,0x48, 0x6f,0x73, 0x74,0x3a, 0x20,0x77, 0x77,0x77, 0x2e,0x61
    .byte 0x64,0x6f, 0x62,0x65, 0x2e,0x63, 0x6f,0x6d, 0x0d,0x0a, 0x43,0x6f, 0x6e,0x6e, 0x65,0x63
    .byte 0x74,0x69, 0x6f,0x6e, 0x3a,0x20, 0x4b,0x65, 0x65,0x70, 0x2d,0x41, 0x6c,0x69, 0x76,0x65
    .byte 0x0d,0x0a, 0x0d,0x0a

packet_404:
    ; ether header, tcp
    .byte 0x00,0x0e, 0x35,0x4c, 0xf7,0x91, 0x00,0x1d, 0xb3,0x84, 0x3f,0x03, 0x08,0x00, 0x45,0x00
    .byte 0x01,0x0a, 0x0e,0x1a, 0x40,0x00, 0xef,0x06, 0x11,0xcf, 0xc0,0x96, 0x12,0x3c, 0xc0,0xa8
    .byte 0xd7,0x89, 0x00,0x50, 0xe9,0xd7, 0xea,0x27, 0xf6,0x52, 0x88,0x10, 0xa0,0x69, 0x50,0x10
    .byte 0x11,0x8a, 0x6b,0x8e, 0x00,0x00
```

```

; payload: "HTTP/1.1 404 Not Found..."
.byte 0x48,0x54, 0x54,0x50, 0x2f,0x31, 0x2e,0x31, 0x20,0x34
.byte 0x30,0x34, 0x20,0x4e, 0x6f,0x74, 0x20,0x46, 0x6f,0x75, 0x6e,0x64, 0x0d,0x0a, 0x44,0x61
.byte 0x74,0x65, 0x3a,0x20, 0x57,0x65, 0x64,0x2c, 0x20,0x31, 0x36,0x20, 0x53,0x65, 0x70,0x20
.byte 0x32,0x30, 0x30,0x39, 0x20,0x32, 0x31,0x3a, 0x34,0x30, 0x3a,0x33, 0x37,0x20, 0x47,0x4d
.byte 0x54,0x0d, 0x0a,0x53, 0x65,0x72, 0x76,0x65, 0x72,0x3a, 0x20,0x41, 0x70,0x61, 0x63,0x68
.byte 0x65,0x0d, 0x0a,0x43, 0x6f,0x6e, 0x74,0x65, 0x6e,0x74, 0x2d,0x54, 0x79,0x70, 0x65,0x3a
.byte 0x20,0x74, 0x65,0x78, 0x74,0x2f, 0x68,0x74, 0x6d,0x6c, 0x0d,0x0a, 0x43,0x61, 0x63,0x68
.byte 0x65,0x2d, 0x43,0x6f, 0x6e,0x74, 0x72,0x6f, 0x6c,0x3a, 0x20,0x6d, 0x61,0x78, 0x2d,0x61
.byte 0x67,0x65, 0x3d,0x39, 0x30,0x30, 0x0d,0x0a, 0x45,0x78, 0x70,0x69, 0x72,0x65, 0x73,0x3a
.byte 0x20,0x57, 0x65,0x64, 0x2c,0x20, 0x31,0x36, 0x20,0x53, 0x65,0x70, 0x20,0x32, 0x30,0x30
.byte 0x39,0x20, 0x32,0x31, 0x3a,0x35, 0x35,0x3a, 0x33,0x37, 0x20,0x47, 0x4d,0x54, 0x0d,0x0a
.byte 0x43,0x6f, 0x6e,0x6e, 0x65,0x63, 0x74,0x69, 0x6f,0x6e, 0x3a,0x20, 0x63,0x6c, 0x6f,0x73
.byte 0x65,0x0d, 0x0a,0x56, 0x61,0x72, 0x79,0x3a, 0x20,0x41, 0x63,0x63, 0x65,0x70, 0x74,0x2d
.byte 0x45,0x6e, 0x63,0x6f, 0x64,0x69, 0x6e,0x67, 0x2c,0x20, 0x55,0x73, 0x65,0x72, 0x2d,0x41
.byte 0x67,0x65, 0x6e,0x74, 0x0d,0x0a, 0x0d,0x0a
packet_arp: ; arp who-has 192.168.215.205 tell 192.168.215.20
.byte 0xff,0xff, 0xff,0xff, 0xff,0xff, 0x08,0x00, 0x0f,0x1e, 0x7e,0x6d, 0x08,0x06, 0x00,0x01
.byte 0x08,0x00, 0x06,0x04, 0x00,0x01, 0x08,0x00, 0x0f,0x1e, 0x7e,0x6d, 0xc0,0xa8, 0xd7,0x14
.byte 0x00,0x00, 0x00,0x00, 0x00,0x00, 0xc0,0xa8, 0xd7,0xcd, 0x00,0x00, 0x00,0x00
.byte 0x00,0x00, 0x00,0x00, 0x00,0x00, 0x00,0x00, 0x01,0xb3, 0xee,0x71
packet_void_udp:
.byte 0x00,0x1d, 0xb3,0x84, 0x3f,0x03, 0x00,0x0e, 0x35,0x4c, 0xf7,0x91, 0x81,0x00, 0xc0,0x00
.byte 0x08,0x00, 0x45,0x00
.byte 0x00,0xc8, 0x00,0x00, 0x40,0x00, 0x40,0x11, 0xe0,0x41, 0xc0,0xa8, 0xd7,0x89, 0xc0,0xa8
.byte 0x01,0x09, 0x1f,0x40, 0x1f,0x4d, 0x00,0xb4, 0x7f,0x5d, 0x80,0x00, 0x82,0x78, 0xb2,0x34
.byte 0x4d,0x69, 0x27,0x29, 0x35,0xf0, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
.byte 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e, 0x7e,0x7e
packet_end:
.section .bss
.byte 0
.section .vectors, "a"
.org 0x1e
.word main

```

NOTES ON WRITING CODE: Code is not real code without comments. You should expect to type twice as many keystrokes for your comments as for your code. If you name your variables properly, you can help the reader understand your code better, and maybe have a few less comments; but this assignment doesn't involve named variables or named subroutines. You have no excuse to not comment your code. The marks for code submitted without comments will be immediately reduced by 50%, and then marked downward from there.

When you copy and paste from a .pdf file, sometimes quote characters are incorrect; Adobe uses fancy quotes that the assembler and C compiler don't like. Look for "a" problems and substitute the quotes with ".

FYI: the prototypes for the above functions are

```

void bwCalculate(void);
int packetFilter(char* packet, int len); // params in registers
char csumBytes(char* packet, int len); // params on stack
int csumWords(char* packet, int len); // params on stack

```