# Examination
# Design of Embedded DSP Processors, TSEA26

| | |
|---|---|
| *Date* | 2012-10-26 |
| *Room* | T2 |
| *Time* | 8-12 |
| *Course code* | TSEA26 |
| *Exam code* | TEN1 |
| *Course name* | Design of Embedded DSP Processors |
| *Department* | ISY |
| *Number of questions* | 5 |
| *Number of pages (including this page)* | 10 |
| *Course responsible* | Andreas Ehliar |
| *Teacher visiting the exam room*<br>*Phone number during the exam time* | Andreas Ehliar |
| *Visiting the exam room* | About 9 och 11 |
| *Course administrator* | Ylva Jernling |
| *Permitted equipment* | None, besides an English dictionary |
| *Grading* | **Points** **Swedish grade**<br>41-50 5<br>31-40 4<br>21-30 3<br>0-20 U |

**Important information:**

- You can answer in English or Swedish.
- **When designing a hardware unit you should attempt to minimize the amount of hardware.** (Unless otherwise noted in the question.)
- The width of data buses and registers must be specified unless otherwise noted. Likewise, the alignment must be specified in all concatenations of signals or buses. When using a box such as *"SATURATE"* or *"ROUND"* in your schematic, you must (unless otherwise noted) describe the content of this box! (E.g. with RTL code). You can assume that all numbers are in two's complement representation unless otherwise noted in the question.
- In questions where you are supposed to write an assembler program based on pseudo code you are allowed to optimize the assembler program in various ways as long as the output of the assembler program is identical to the output from the pseudo code. You can also (unless otherwise noted in the question) assume that hazards will not occur due to parts of the processor that you are not designing.
- Premade control tables are available at the end of the exam if you prefer that to drawing your own.

**Good luck!**

## Question 1: Address Generator Unit(8p)

Draw a schematic and a control table for an AGU with the following specifications:

- It should have two address registers
- It should have a post-increment addressing mode with a step size of one
- It should have a pre-decrement addressing mode with a step size of one
- It should have a modulo addressing mode with a step size of one. (Note: Modulo addressing is sometimes called circular addressing as well.)
- It should be possible to set the address registers (and all other registers necessary to support these addressing modes) to arbitrary values.
- There should be one 16-bit wide input to the AGU which comes from the register file.
- It should have one 16-bit wide address output to the data memory.

## Question 2: Arithmetic Logic Unit(13p)

The following is a list of desired ALU operations:

- RESULT = OpA + OpB
- RESULT = OpA - OpB
- RESULT = SAT(OpA + OpB)
- RESULT = SAT(OpA - OpB)
- RESULT = (OpA + OpB)/2
- RESULT = SAT(ABS(OpA) - ABS(OpB))
- RESULT = SAT(ABS(OpA - OpB))
- RESULT = SAT(ABS(OpA))

**Constraints**: OpA and OpB comes from the register file and are 16 bits wide. RESULT should also be 16 bits wide.

However, it turns out that it is desirable to implement an ALU where only one adder is used (for area, power, and critical path purposes). This means that some of the operations listed above cannot be implemented in a single clock cycle.

*Note: If you choose to answer both part a and part b you only have to draw one schematic/control table.*

**Tasks:**

(a) (7p) Analyze the list of ALU instructions given above and determine which instructions that can be implemented under the constraint that only one adder can be used in the ALU. Draw a schematic and a control table for an ALU that supports these instructions. (Note: Only single cycle instructions are allowed.)

(b) (6p) Since the remaining operations are also desirable, you need to figure out a way to perform these operations in at most two clock cycles. In order to do that you may need to add new instructions to your ALU and perhaps new flags and/or registers. Determine whether you need to add any new instructions to your ALU and modify your schematic and control table so that those new instructions are supported. Finally you need to describe how these new instructions allows you to perform the operations that you couldn't include in part a. *Important:* You are not allowed to introduce multi-cycle instructions!

## Question 3: Program Flow Control Unit(7p)

The following two assembler level functions has been written and your task is to create a PFC unit that can support these functions. (We assume that your DSP processor is a normal in-order single issue RISC-like processor like the Senior processor.)

```
function1:
    set r15,#32
    move ar0,r1
    move ar1,r2
    clear acr0
loop:
    mac acr0, DM0[ar0++],DM1[ar1++]
    add r15,r15,#-1
    bne loop
    nop          // Delay slot
    nop          // Delay slot

    rts
    satrnd       r0,acr0 // Delay slot

function2:
    set r14,#128
loop2:
    jsr function1
    nop          // Delay slot

    store DM0[ar2++],r0
    add   r1,#1
    add   r14,r14,#-1
    bne   loop2
    nop   // Delay slot
    nop   // Delay slot

    rts
    nop   // Delay slot
```

**Inputs and outputs:**

- Z flag input from the ALU (1 bit wide)
- Address output to the program memory (16 bit wide)
- And of course your control signals

**Tasks:**

(a) (5p) Implement a PFC unit that supports the PFC operations given in the assembler program listed above. A hardware stack with at least two levels must be used. You need to include a schematic and a control table for your PFC unit. You don't need to worry about the pipeline in this exercise and you can assume that the instruction decoder will send the correct operation at the correct time.

(b) (2p) Rewrite function1 so that the NOPs in the delay slots are eliminated. (The program should of course produce the same result in r0 after the rewrite.)

## Question 4: MAC UNIT(15p)

Design a MAC unit capable of supporting the following functions:

```
function save_state(ptr)
  // This function should save all of the state in the MAC unit to
  // memory. (The context for this would be an interrupt handler in a
  // multi-tasking operating system.) Should take less than 32 cycles.
endfunction

function restore_state(ptr)
  // Same as save_state but this one should restore all state saved
  // by save_state(). Should take less than 32 cycles.
endfunction

// The parameters to this function are passed in register r0-r5.
// This function should use at most 150 clock cycles.
function matrix_x_vectors(A, B, C, D, vecptr, resultptr)
  repeat 128
    tmp = DM0[vecptr++]

    even = signextend(tmp[15:0])
    odd  = signextend(tmp[31:16])

    evenresult = even * A + odd * B
    evenresult = evenresult + 0x4000

    if (evenresult >= 0x40000000) then
      evenresult = 0x3fffffff
    endif
    if(evenresult < -0x40000000) then
      evenresult = -0x40000000
    endif

    oddresult = even * C + odd * D
    oddresult = oddresult + 0x4000

    if (oddresult >= 0x40000000) then
      oddresult = 0x3fffffff
    endif
    if(oddresult < -0x40000000) then
      oddresult = -0x40000000
    endif

    tmp[15:0] = evenresult[30:15]
    tmp[31:0] = oddresult[30:15]
    DM1[resultptr++] = tmp
  endrepeat

endfunction
```

```
// The parameters are passed in register r0 and r1
// This function should use at most 40 clock cycles
function cplx_dotproduct(sampleptr, coeffptr)
  sum_real = 0
  sum_imag = 0

  repeat 30
    sample = DM0[sampleptr++]
    coeff = DM1[coeffptr++]
    sample_real = signextend(sample[15:0])
    sample_imag = signextend(sample[31:16])
    coeff_real = signextend(coeff[15:0])
    coeff_imag = signextend(coeff[31:16])

    sum_real = sum_real + sample_real * coeff_real - sample_imag * coeff_imag
    sum_imag = sum_imag + sample_real * coeff_imag + sample_imag * coeff_real
  endrepeat

  if(sum_real >= 0x80000000) then
     sum_real = 0x7fffffff
  endif
  if(sum_real <= -0x80000000) then
     sum_real = -0x80000000
  endif
  if(sum_imag >= 0x80000000) then
     sum_imag = 0x7fffffff
  endif
  if(sum_imag <= -0x80000000) then
     sum_imag = -0x80000000
  endif

  r0 = sum_real[31:16]  // Return the result in register r0
  r1 = sum_imag[31:16]  // and register r1
endfunction
```

- These are the allowed inputs and outputs to your module:
  - From_DM0 and From_DM1: 32 bit wide input from data memory 0 and 1.
  - OpA/OpB: 16 bit wide inputs from the register file
  - To_RF: 16 bit wide output to the writeback port of the register file
  - TO_DM1: 32 bit wide output to data memory 1
- Your accumulator registers should be 40 bit wide.
- You are allowed to use up to four multipliers in your MAC unit.

**Tasks:**

(a) (6p) Decide on an instruction set for your MAC unit and write assembly code for all functions listed above. If you don't answer part b you need to carefully specify what each of your instructions does.

(b) (9p) Draw a schematic and a control table for the MAC unit

## Question 5: Misc knowledge(7p)

Each of the questions should be answered fairly briefly using *at most* five sentences per question. You are encouraged to draw a figure if it will simplify or clarify your explanation.

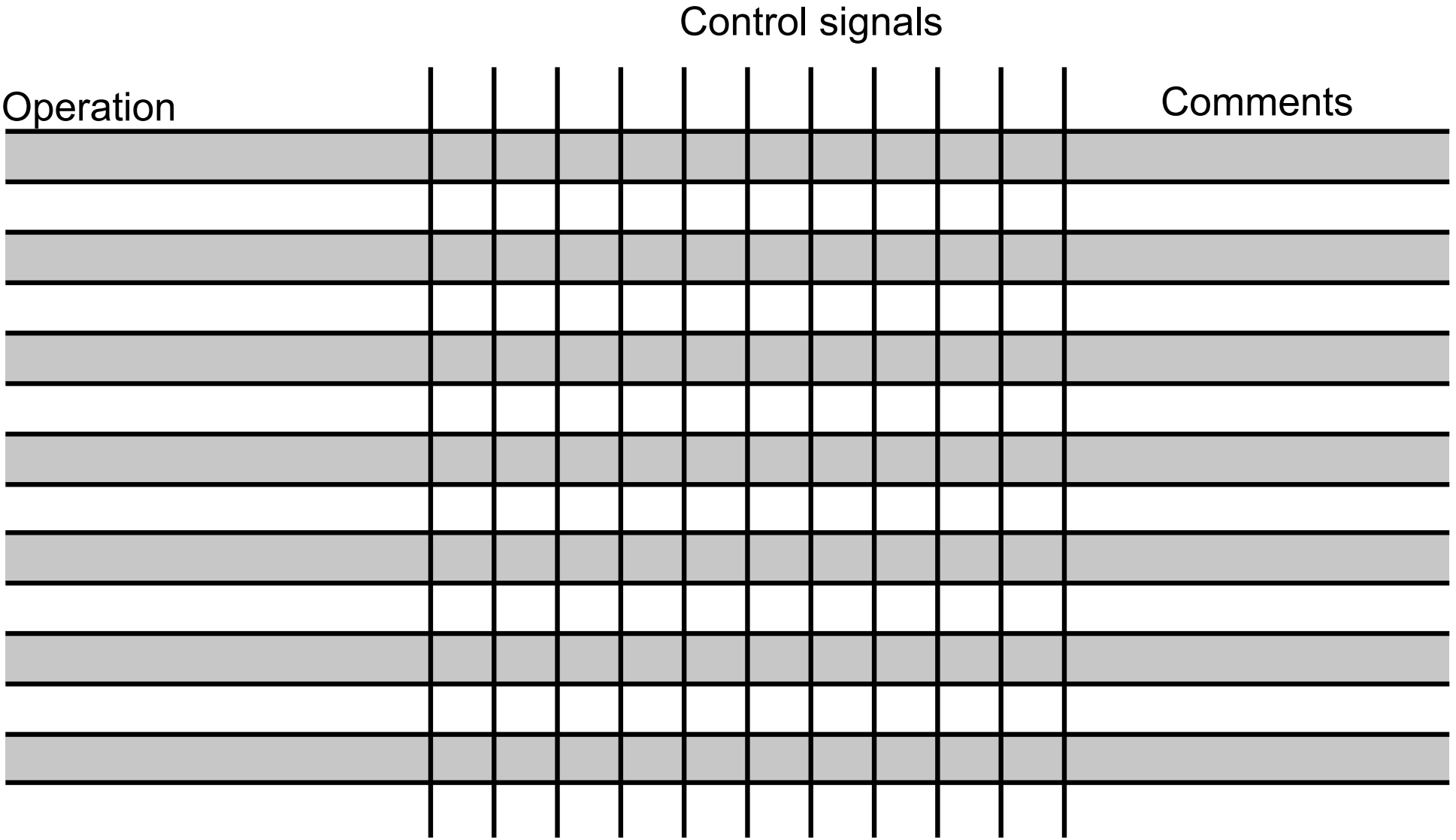
(a) (3p) Explain briefly the concept of profiling and how you would want to use this in the context of ASIP development.

(b) (2p) Discuss at least one advantages and one disadvantage of using floating point arithmetic in a DSP processor instead of fixed point arithmetic.

(c) (2p) Explain briefly the concept of dynamic scaling (which is sometimes called block floating point).

# Control table for Q1 (AGU)

Feel free to use this table when answering question 1. Remove it from the exam and hand it in together with the rest of the exam in that case. (Or draw your own control table if you prefer.)




AID: _____ Date: 2012-10-26. Page number _____

Course code: TSEA26 Exam code: TEN1

# Control table for Q2 (ALU)

Feel free to use this table when answering question 2. Remove it from the exam and hand it in together with the rest of the exam in that case. (Or draw your own control table if you prefer.)

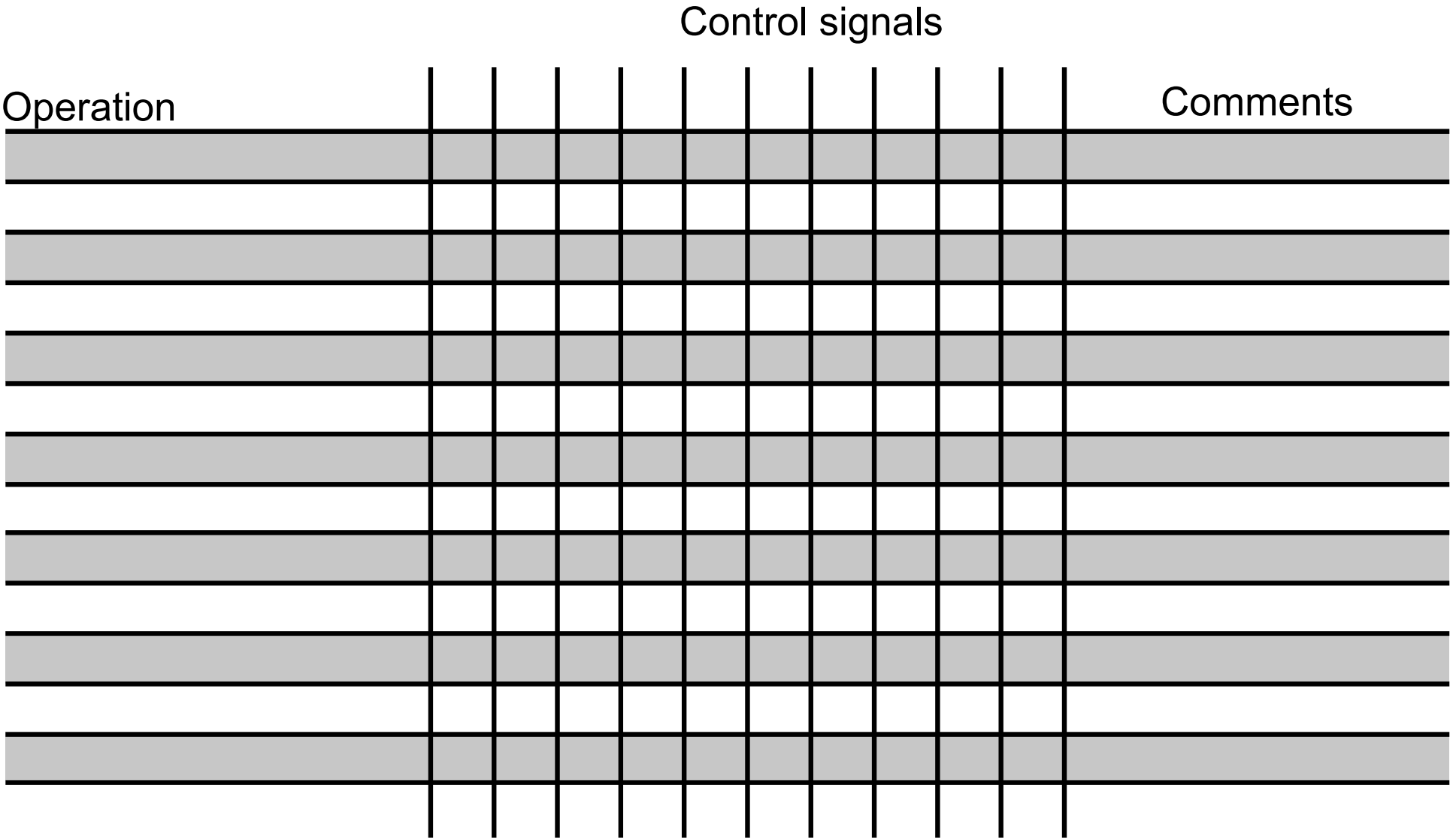
| Operation | Control signals | | | | | | | | Comments |
|---|---|---|---|---|---|---|---|---|---|
| OpA + OpB | | | | | | | | | |
| OpA - OpB | | | | | | | | | |
| SAT(OpA + OpB) | | | | | | | | | |
| SAT(OpA - OpB) | | | | | | | | | |
| (OpA + OpB) / 2 | | | | | | | | | |
| SAT(ABS(OpA) - ABS(OpB)) | | | | | | | | | |
| SAT(ABS(OpA - OpB)) | | | | | | | | | |
| SAT(ABS(OpA)) | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |
| | | | | | | | | | |

AID: _____ Date: 2012-10-26. Page number _____

Course code: TSEA26 Exam code: TEN1

# Control table for Q3 (PFC)

Feel free to use this table when answering question 3. Remove it from the exam and hand it in together with the rest of the exam in that case. (Or draw your own control table if you prefer.)

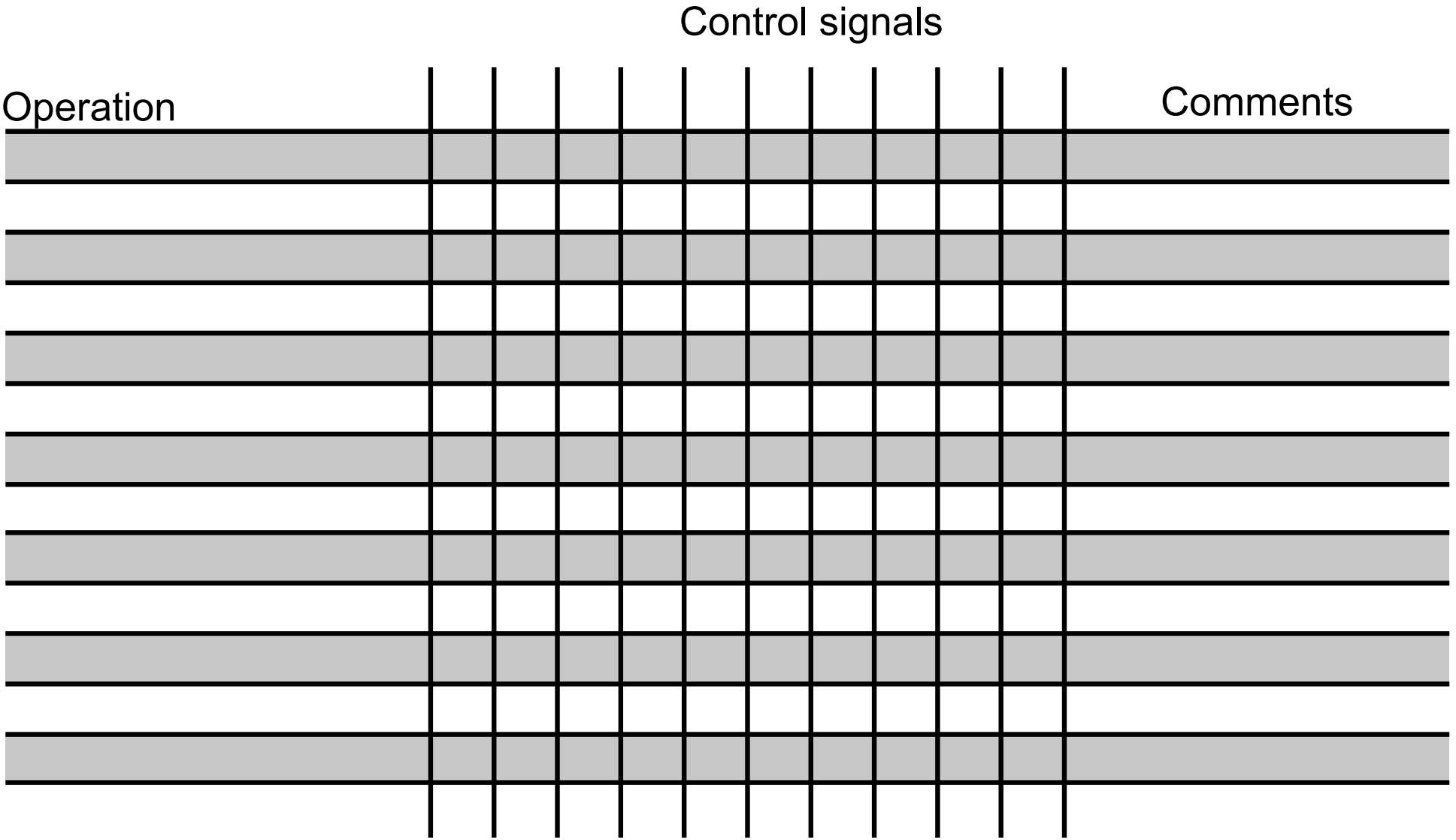



AID: _____ Date: 2012-10-26. Page number _____

Course code: TSEA26 Exam code: TEN1

# Control table for Q4 (MAC)

Feel free to use this table when answering question 4. Remove it from the exam and hand it in together with the rest of the exam in that case. (Or draw your own control table if you prefer.)




AID: _____ Date: 2012-10-26. Page number _____

Course code: TSEA26 Exam code: TEN1