

Final Exam S3

Computer Architecture

Duration: 1 hr 30 min

Write answers only on the answer sheet.

Do not use a pencil or red ink.

Exercise 1 (3 points)

Complete the table shown on the [answer sheet](#). Write down the new values of the registers (except the PC) and memory that are modified by the instructions. **Use the hexadecimal representation. Memory and registers are reset to their initial values for each instruction.**

Initial values: D0 = \$FFFF0005 A0 = \$00005000 PC = \$00006000
 D1 = \$FFFFFFF2 A1 = \$00005008
 D2 = \$FFFF0002 A2 = \$00005010

| | |
|----------|-------------------------|
| \$005000 | 54 AF 18 B9 E7 21 48 C0 |
| \$005008 | C9 10 11 C8 D4 36 1F 88 |
| \$005010 | 13 79 01 80 42 1A 2D 49 |

Exercise 2 (2 points)

Complete the table shown on the [answer sheet](#). Give the result of the additions and the values of the N, Z, V and C flags.

Exercise 3 (3 points)

Let us consider the following program. Complete the table shown on the [answer sheet](#).

```

Main      move.l  #fff0,d7
next1     moveq.l #1,d1
           cmpi.l  #$1000,d7
           bgt   next2
           moveq.l #2,d1
next2     clr.l   d2
           move.l #200,d0
loop2     addq.l #1,d2
           subq.b #4,d0
           bne   loop2
next3     clr.l   d3
           move.l #$77777777,d0
loop3     addq.l #1,d3
           dbra  d0,loop3      ; DBRA = DBF

```

Exercise 4 (2 points)

Write the instructions that modify D1 so that it takes the value given on the [answer sheet](#). The initial value of D1 is \$ 87654321 . **Use the SWAP, ROR and ROL instructions only.** Answer on the [answer sheet](#).

Exercise 5 (10 points)

All the questions in this exercise are independent. **Except for the output registers, none of the data or address registers must be modified when the subroutine returns.** A string of characters always ends with a null character (the value zero). For the whole exercise, we assume that the strings of characters are never empty (they contain at least one character different from the null character). An array of strings is made up of multiple strings in a row. For the whole exercise, we assume that an array of strings always contains at least one non-empty string. An array of strings always ends with two zeros: the null character of the last string followed by an additional final zero that marks the end of the array.

1. Write the **next_str** subroutine that returns the address of the next string in an array of strings (or the last zero in the array if there are no more strings).

Input: **A0.L** points to a string in the array.

Output: **A0.L** points to the next string in the array or the last zero in the array if there are no more strings.

Be careful. The next_str subroutine must contain 3 lines of instructions at the most.

2. Write the **two_by_two_swap** subroutine that swaps the characters of a string in pairs (for odd lengths, the last character does not change).

Input: **A0.L** points to a string of characters.

Output: The string is modified in place (directly in memory).

For instance:

- If **A0.L** points to the string “ABCDEF”, the string will become “BADCFE”.
- If **A0.L** points to the string “ABCDEFG”, the string will become “BADCFEG”.

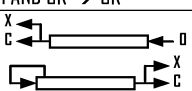
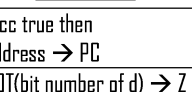
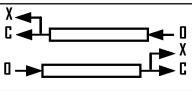
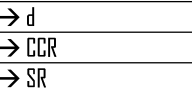
Be careful. The two_by_two_swap subroutine must contain 13 lines of instructions at the most.

3. By using the **next_str** and **two_by_two_swap** subroutines, write the **swap_all** subroutine that swap by pairs the characters of all the strings in an array of strings. If a string contains an odd number of characters, its last character does not change.

Input: **A0.L** points to the first string in an array of strings.

Output: All of the strings are modified in place.

Be careful. The swap_all subroutine must contain 10 lines of instructions at the most.

| Opcode | Size | Operand | CCR | Effective Address s=source, d=destination, e=either, i=displacement | | | | | | | | | | | Operation | Description | |
|-------------------|-----------------|-------------------------|---------|---|----------------|------|-------|-------|--------|-----------|-------|-------|--------|-----------|-----------|--|--|
| | BWL | s,d | XNZVC | Dn | An | (An) | (An)+ | -(An) | (i.An) | (i.An,Rn) | abs.W | abs.L | (i.PC) | (i.PC,Rn) | #n | | |
| ABCD | B | Dy,Dx -(Ay),-(Ax) | *U*U* | e | - | - | - | - | - | - | - | - | - | - | - | $Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$ $-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$ | Add BCD source and eXtend bit to destination, BCD result |
| ADD ⁴ | BWL | s,Dn Dn,d | ***** | e | s ⁴ | s | s | s | s | s | s | s | s | s | s | $s + Dn \rightarrow Dn$ $Dn + d \rightarrow d$ | Add binary (ADDI or ADDQ is used when source is #n. Prevent ADDQ with #n.L) |
| ADDA ⁴ | WL | s,An | ----- | s | e | s | s | s | s | s | s | s | s | s | s | $s + An \rightarrow An$ | Add address (.W sign-extended to .L) |
| ADDI ⁴ | BWL | #n,d | ***** | d | - | d | d | d | d | d | d | d | - | - | - | $#n + d \rightarrow d$ | Add immediate to destination |
| ADDQ ⁴ | BWL | #n,d | ***** | d | d | d | d | d | d | d | d | d | - | - | - | $#n + d \rightarrow d$ | Add quick immediate (#n range: 1 to 8) |
| ADDX | BWL | Dy,Dx -(Ay),-(Ax) | ***** | e | - | - | - | - | - | - | - | - | - | - | - | $Dy + Dx + X \rightarrow Dx$ $-(Ay) + -(Ax) + X \rightarrow -(Ax)$ | Add source and eXtend bit to destination |
| AND ⁴ | BWL | s,Dn Dn,d | -**00 | e | - | s | s | s | s | s | s | s | s | s | s | $s \text{ AND } Dn \rightarrow Dn$ $Dn \text{ AND } d \rightarrow d$ | Logical AND source to destination (ANDI is used when source is #n) |
| ANDI ⁴ | BWL | #n,d | -**00 | d | - | d | d | d | d | d | d | d | - | - | - | $#n \text{ AND } d \rightarrow d$ | Logical AND immediate to destination |
| ANDI ⁴ | B | #n,CCR | ===== | - | - | - | - | - | - | - | - | - | - | - | - | $#n \text{ AND } CCR \rightarrow CCR$ | Logical AND immediate to CCR |
| ANDI ⁴ | W | #n,SR | ===== | - | - | - | - | - | - | - | - | - | - | - | - | $#n \text{ AND } SR \rightarrow SR$ | Logical AND immediate to SR (Privileged) |
| ASL | BWL | Dx,Dy | ***** | e | - | - | - | - | - | - | - | - | - | - | - |  | Arithmetic shift Dy by Dx bits left/right |
| ASR | W | #n,Dy d | ***** | d | - | - | - | - | - | - | - | - | - | - | - |  | Arithmetic shift Dy #n bits L/R (#n: 1 to 8) Arithmetic shift ds 1 bit left/right (.W only) |
| Bcc | BW ³ | address ² | ----- | - | - | - | - | - | - | - | - | - | - | - | - | if cc true then address \rightarrow PC | Branch conditionally (cc table on back) (8 or 16-bit \pm offset to address) |
| BCHG | B L | Dn,d #n,d | ---*--- | e ¹ | - | d | d | d | d | d | d | d | - | - | - | $\text{NOT}(\text{bit number of } d) \rightarrow Z$ $\text{NOT}(\text{bit } n \text{ of } d) \rightarrow \text{bit } n \text{ of } d$ | Set Z with state of specified bit in d then invert the bit in d |
| BCLR | B L | #n,d #n,d | ---*--- | e ¹ | - | d | d | d | d | d | d | d | - | - | - | $\text{NOT}(\text{bit number of } d) \rightarrow Z$ $0 \rightarrow \text{bit number of } d$ | Set Z with state of specified bit in d then clear the bit in d |
| BRA | BW ³ | address ² | ----- | - | - | - | - | - | - | - | - | - | - | - | - | address \rightarrow PC | Branch always (8 or 16-bit \pm offset to addr) |
| BSET | B L | Dn,d #n,d | ---*--- | e ¹ | - | d | d | d | d | d | d | d | - | - | - | $\text{NOT}(\text{bit } n \text{ of } d) \rightarrow Z$ $1 \rightarrow \text{bit } n \text{ of } d$ | Set Z with state of specified bit in d then set the bit in d |
| BSR | BW ³ | address ² | ----- | - | - | - | - | - | - | - | - | - | - | - | - | PC \rightarrow -(SP); address \rightarrow PC | Branch to subroutine (8 or 16-bit \pm offset) |
| BTST | B L | Dn,d #n,d | ---*--- | e ¹ | - | d | d | d | d | d | d | d | d | d | d | $\text{NOT}(\text{bit } Dn \text{ of } d) \rightarrow Z$ $\text{NOT}(\text{bit } \#n \text{ of } d) \rightarrow Z$ | Set Z with state of specified bit in d Leave the bit in d unchanged |
| CHK | W | s,Dn | -*UUU | e | - | s | s | s | s | s | s | s | s | s | s | if $Dn < 0$ or $Dn > s$ then TRAP | Compare Dn with 0 and upper bound [s] |
| CLR | BWL | d | -0100 | d | - | d | d | d | d | d | d | d | - | - | - | $0 \rightarrow d$ | Clear destination to zero |
| CMP ⁴ | BWL | s,Dn | -***** | e | s ⁴ | s | s | s | s | s | s | s | s | s | s | set CCR with $Dn - s$ | Compare Dn to source |
| CMPA ⁴ | WL | s,An | -***** | s | e | s | s | s | s | s | s | s | s | s | s | set CCR with $An - s$ | Compare An to source |
| CMPI ⁴ | BWL | #n,d | -***** | d | - | d | d | d | d | d | d | d | - | - | - | set CCR with $d - \#n$ | Compare destination to #n |
| CMPM ⁴ | BWL | (Ay)+,(Ax)+ | -***** | - | - | - | e | - | - | - | - | - | - | - | - | set CCR with $(Ax) - (Ay)$ | Compare (Ax) to (Ay); Increment Ax and Ay |
| DBcc | W | Dn,address ² | ----- | - | - | - | - | - | - | - | - | - | - | - | - | if cc false then { $Dn - 1 \rightarrow Dn$ if $Dn < -1$ then addr \rightarrow PC } | Test condition, decrement and branch (16-bit \pm offset to address) |
| DIVS | W | s,Dn | -***0 | e | - | s | s | s | s | s | s | s | s | s | s | $\pm 32\text{bit } Dn / \pm 16\text{bit } s \rightarrow \pm Dn$ | $Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$ |
| DIVU | W | s,Dn | -***0 | e | - | s | s | s | s | s | s | s | s | s | s | $32\text{bit } Dn / 16\text{bit } s \rightarrow Dn$ | $Dn = [16\text{-bit remainder}, 16\text{-bit quotient}]$ |
| EOR ⁴ | BWL | Dn,d | -**00 | e | - | d | d | d | d | d | d | d | - | - | - | $Dn \text{ XOR } d \rightarrow d$ | Logical exclusive OR Dn to destination |
| EORI ⁴ | BWL | #n,d | -**00 | d | - | d | d | d | d | d | d | d | - | - | - | $\#n \text{ XOR } d \rightarrow d$ | Logical exclusive OR #n to destination |
| EORI ⁴ | B | #n,CCR | ===== | - | - | - | - | - | - | - | - | - | - | - | - | $\#n \text{ XOR } CCR \rightarrow CCR$ | Logical exclusive OR #n to CCR |
| EORI ⁴ | W | #n,SR | ===== | - | - | - | - | - | - | - | - | - | - | - | - | $\#n \text{ XOR } SR \rightarrow SR$ | Logical exclusive OR #n to SR (Privileged) |
| EXG | L | Rx,Ry | ----- | e | e | - | - | - | - | - | - | - | - | - | - | register \leftrightarrow register | Exchange registers (32-bit only) |
| EXT | WL | Dn | -**00 | d | - | - | - | - | - | - | - | - | - | - | - | $Dn.B \rightarrow Dn.W \mid Dn.W \rightarrow Dn.L$ | Sign extend (change .B to .W or .W to .L) |
| ILLEGAL | | | ----- | - | - | - | - | - | - | - | - | - | - | - | - | PC \rightarrow -(SSP); SR \rightarrow -(SSP) | Generate Illegal Instruction exception |
| JMP | | d | ----- | - | - | d | - | - | d | d | d | d | d | d | d | $\uparrow d \rightarrow PC$ | Jump to effective address of destination |
| JSR | | d | ----- | - | - | d | - | - | d | d | d | d | d | d | d | PC \rightarrow -(SP); $\uparrow d \rightarrow PC$ | push PC, jump to subroutine at address d |
| LEA | L | s,An | ----- | - | e | s | - | - | s | s | s | s | s | s | s | $\uparrow s \rightarrow An$ | Load effective address of s to An |
| LINK | | An,#n | ----- | - | - | - | - | - | - | - | - | - | - | - | - | $An \rightarrow -(SP); SP \rightarrow An;$ $SP + \#n \rightarrow SP$ | Create local workspace on stack (negative n to allocate space) |
| LSL | BWL | Dx,Dy | ***0* | e | - | - | - | - | - | - | - | - | - | - | - |  | Logical shift Dy, Dx bits left/right |
| LSR | W | #n,Dy d | ***0* | d | - | - | - | - | - | - | - | - | - | - | - |  | Logical shift Dy, #n bits L/R (#n: 1 to 8) Logical shift d 1 bit left/right (.W only) |
| MOVE ⁴ | BWL | s,d | -**00 | e | s ⁴ | e | e | e | e | e | e | e | s | s | s | $s \rightarrow d$ | Move data from source to destination |
| MOVE | W | s,CCR | ===== | s | - | s | s | s | s | s | s | s | s | s | s | $s \rightarrow CCR$ | Move source to Condition Code Register |
| MOVE | W | s,SR | ===== | s | - | s | s | s | s | s | s | s | s | s | s | $s \rightarrow SR$ | Move source to Status Register (Privileged) |
| MOVE | W | SR,d | ----- | d | - | d | d | d | d | d | d | d | - | - | - | $SR \rightarrow d$ | Move Status Register to destination |
| MOVE | L | USP,An An,USP | ----- | - | d | - | - | - | - | - | - | - | - | - | - | $USP \rightarrow An$ $An \rightarrow USP$ | Move User Stack Pointer to An (Privileged) Move An to User Stack Pointer (Privileged) |
| | BWL | s,d | XNZVC | Dn | An | (An) | (An)+ | -(An) | (i.An) | (i.An,Rn) | abs.W | abs.L | (i.PC) | (i.PC,Rn) | #n | | |

Computer Architecture – EPITA – S3 – 2023/2024

| Opcode | Size | Operand | CCR | Effective Address | s=source, d=destination, e=either, i=displacement | | | | | | | | | | | Operation | Description | | |
|--------------------|------|------------------------|--------|---|---|-----------------|----------------|---|--|--|--|--|--|--|--|-----------|-------------|--|--|
| | BWL | s,d | XNZVC | Dn An (An) (An)+ -(An) (i.An) (i.An,Rn) | abs.W abs.L (i.PC) (i.PC,Rn) | #n | | | | | | | | | | | | | |
| MOVEA ⁴ | WL | s,An | ----- | s e s s s s s s | s s s s s s s s | s s s s s s s s | s ⁴ | s → An | Move source to An (MOVE s,An use MOVEA) | | | | | | | | | | |
| MOVEM ⁴ | WL | Rn-Rn,d s,Rn-Rn | ----- | - - d - d d d d | - - s s - s s s | - - - - - - - - | - - | Registers → d s → Registers | Move specified registers to/from memory (W source is sign-extended to .L for Rn) | | | | | | | | | | |
| MOVEP | WL | Dn,(i.An) (i.An),Dn | ----- | s - - - - d - | - - - - - s - - | - - - - - - - - | - - | Dn → (i.An)...(i+2,An)...(i+4,A. (i.An) → Dn...(i+2,An)...(i+4,A. | Move Dn to/from alternate memory bytes (Access only even or odd addresses) | | | | | | | | | | |
| MOVEQ ⁴ | L | #n,Dn | -***00 | d - - - - - | - - - - - - - - | - - - - - - - - | s | #n → Dn | Move sign extended 8-bit #n to Dn | | | | | | | | | | |
| MULS | W | s,Dn | -***00 | e - s s s s s s s s | s s s s s s s s | s s s s s s s s | s | ±16bit s * ±16bit Dn → ±Dn | Multiply signed 16-bit; result: signed 32-bit | | | | | | | | | | |
| MULU | W | s,Dn | -***00 | e - s s s s s s s s | s s s s s s s s | s s s s s s s s | s | 16bit s * 16bit Dn → Dn | Multiply unisig'd 16-bit; result: unisig'd 32-bit | | | | | | | | | | |
| NBCD | B | d | *U*U* | d - d d d d d d d d | d d d d d d d d | - - - - - - - - | - | 0 - d ₁₀ - X → d | Negate BCD with eXtend, BCD result | | | | | | | | | | |
| NEG | BWL | d | ***** | d - d d d d d d d d | d d d d d d d d | - - - - - - - - | - | 0 - d → d | Negate destination (2's complement) | | | | | | | | | | |
| NEGX | BWL | d | ***** | d - d d d d d d d d | d d d d d d d d | - - - - - - - - | - | 0 - d - X → d | Negate destination with eXtend | | | | | | | | | | |
| NDP | | | ----- | - - - - - | - - - - - | - - - - - | - | None | No operation occurs | | | | | | | | | | |
| NOT | BWL | d | -***00 | d - d d d d d d d d | d d d d d d d d | - - - - - - - - | - | NOT(d) → d | Logical NOT destination (1's complement) | | | | | | | | | | |
| OR ⁴ | BWL | s,Dn Dn,d | -***00 | e - s s s s s s s s | s s s s s s s s | s s s s s s s s | s ⁴ | s OR Dn → Dn Dn OR d → d | Logical OR (ORI is used when source is #n) | | | | | | | | | | |
| ORI ⁴ | BWL | #n,d | -***00 | d - d d d d d d d d | d d d d d d d d | - - - - - - - - | s | #n OR d → d | Logical OR #n to destination | | | | | | | | | | |
| ORI ⁴ | B | #n,CCR | ===== | - - - - - | - - - - - | - - - - - | - | #n OR CCR → CCR | Logical OR #n to CCR | | | | | | | | | | |
| ORI ⁴ | W | #n,SR | ===== | - - - - - | - - - - - | - - - - - | - | #n OR SR → SR | Logical OR #n to SR (Privileged) | | | | | | | | | | |
| PEA | L | s | ----- | - - s - - | - s s s s s s s s | - - - - - - - - | - | ↑s → -(SP) | Push effective address of s onto stack | | | | | | | | | | |
| RESET | | | ----- | - - - - - | - - - - - | - - - - - | - | Assert RESET Line | Issue a hardware RESET (Privileged) | | | | | | | | | | |
| ROL | BWL | Dx,Dy #n,Dy | -***0* | e - - - - - | - - - - - | - - - - - | - | | Rotate Dy, Dx bits left/right (without X) | | | | | | | | | | |
| ROR | W | d | -***0* | d - - - - - | - - - - - | - - - - - | - | | Rotate Dy, #n bits left/right (#n: 1 to 8) | | | | | | | | | | |
| ROXL | BWL | Dx,Dy #n,Dy | ***0* | e - - - - - | - - - - - | - - - - - | - | | Rotate Dy, Dx bits L/R, X used then updated | | | | | | | | | | |
| ROXR | W | d | ***0* | d - - - - - | - - - - - | - - - - - | - | | Rotate Dy, #n bits left/right (#n: 1 to 8) | | | | | | | | | | |
| RTE | | | ===== | - - - - - | - - - - - | - - - - - | - | (SP)+ → SR; (SP)+ → PC | Return from exception (Privileged) | | | | | | | | | | |
| RTR | | | ===== | - - - - - | - - - - - | - - - - - | - | (SP)+ → CCR; (SP)+ → PC | Return from subroutine and restore CCR | | | | | | | | | | |
| RTS | | | ----- | - - - - - | - - - - - | - - - - - | - | (SP)+ → PC | Return from subroutine | | | | | | | | | | |
| SBCD | B | Dy,Dx (-Ay),(-Ax) | *U*U* | e - - - - - | - - - - - | - - - - - | - | Dx ₁₀ - Dy ₁₀ - X → Dx ₁₀ (-Ax) ₁₀ - (-Ay) ₁₀ - X → (-Ax) ₁₀ | Subtract BCD source and eXtend bit from destination, BCD result | | | | | | | | | | |
| Scc | B | d | ----- | d - d d d d d d d d | d d d d d d d d | - - - - - - - - | - | If cc is true then 1's → d else 0's → d | If cc true then d.B = 11111111 else d.B = 00000000 | | | | | | | | | | |
| STOP | | #n | ===== | - - - - - | - - - - - | - - - - - | - | #n → SR; STOP | Move #n to SR, stop processor (Privileged) | | | | | | | | | | |
| SUB ⁴ | BWL | s,Dn Dn,d | ***** | e s s s s s s s s | s s s s s s s s | s s s s s s s s | s ⁴ | Dn - s → Dn d - Dn → d | Subtract binary (SUBI or SUBQ used when source is #n. Prevent SUBQ with #n.L) | | | | | | | | | | |
| SUBA ⁴ | WL | s,An | ----- | s e s s s s s s s s | s s s s s s s s | s s s s s s s s | s | An - s → An | Subtract address (.W sign-extended to .L) | | | | | | | | | | |
| SUBI ⁴ | BWL | #n,d | ***** | d - d d d d d d d d | d d d d d d d d | - - - - - - - - | s | d - #n → d | Subtract immediate from destination | | | | | | | | | | |
| SUBQ ⁴ | BWL | #n,d | ***** | d d d d d d d d | d d d d d d d d | - - - - - - - - | s | d - #n → d | Subtract quick immediate (#n range: 1 to 8) | | | | | | | | | | |
| SUBX | BWL | Dy,Dx (-Ay),(-Ax) | ***** | e - - - - - | - - - - - | - - - - - | - | Dx - Dy - X → Dx (-Ax) - (-Ay) - X → (-Ax) | Subtract source and eXtend bit from destination | | | | | | | | | | |
| SWAP | W | Dn | -***00 | d - - - - - | - - - - - | - - - - - | - | bits[31:16] ↔ bits[15:0] | Exchange the 16-bit halves of Dn | | | | | | | | | | |
| TAS | B | d | -***00 | d - d d d d d d d d | d d d d d d d d | - - - - - - - - | - | test d → CCR; 1 → bit7 of d | N and Z set to reflect d, bit7 of d set to 1 | | | | | | | | | | |
| TRAP | | #n | ----- | - - - - - | - - - - - | - - - - - | s | PC → -(SSP); SR → -(SSP); (vector table entry) → PC | Push PC and SR, PC set by vector table #n (#n range: 0 to 15) | | | | | | | | | | |
| TRAPV | | | ----- | - - - - - | - - - - - | - - - - - | - | If V then TRAP #7 | If overflow, execute an Overflow TRAP | | | | | | | | | | |
| TST | BWL | d | -***00 | d - d d d d d d d d | d d d d d d d d | - - - - - - - - | - | test d → CCR | N and Z set to reflect destination | | | | | | | | | | |
| UNLK | BWL | An | ----- | - d - - - | - - - - - | - - - - - | - | An → SP; (SP)+ → An | Remove local workspace from stack | | | | | | | | | | |
| | BWL | s,d | XNZVC | Dn An (An) (An)+ -(An) (i.An) (i.An,Rn) | abs.W abs.L (i.PC) (i.PC,Rn) | #n | | | | | | | | | | | | | |

| Condition Tests (+ OR, ! NOT, ⊕ XOR, * Unsigned, # Alternate cc) | | | | | |
|--|----------------|----------|----|------------------|--------------|
| cc | Condition | Test | cc | Condition | Test |
| T | true | 1 | VC | overflow clear | !V |
| F | false | 0 | VS | overflow set | V |
| HI* | higher than | !(C + Z) | PL | plus | !N |
| LS* | lower or same | C + Z | MI | minus | N |
| HS*, CC* | higher or same | !C | GE | greater or equal | !(N ⊕ V) |
| LD*, CS* | lower than | C | LT | less than | (N ⊕ V) |
| NE | not equal | !Z | GT | greater than | !(N ⊕ V) + Z |
| EQ | equal | Z | LE | less or equal | (N ⊕ V) + Z |

An Address register (16/32-bit, n=0-7)
Dn Data register (8/16/32-bit, n=0-7)
Rn any data or address register
s Source, d Destination
e Either source or destination
#n Immediate data, i Displacement
BCD Binary Coded Decimal
↑ Effective address
1 Long only; all others are byte only
2 Assembler calculates offset
3 Branch sizes: .B or .S -128 to +127 bytes, .W or .L -32768 to +32767 bytes
4 Assembler automatically uses A, I, Q or M form if possible. Use #n.L to prevent Quick optimization

SSP Supervisor Stack Pointer (32-bit)
USP User Stack Pointer (32-bit)
SP Active Stack Pointer (same as A7)
PC Program Counter (24-bit)
SR Status Register (16-bit)
CCR Condition Code Register (lower 8-bits of SR)
N negative, Z zero, V overflow, C carry, X extend
* set according to operation's result, = set directly
- not affected, 0 cleared, 1 set, U undefined

Revised by Peter Csaszar, Lawrence Tech University – 2004-2006

Distributed under the GNU general public use license.

Last name: First name: Group:

ANSWER SHEET TO BE HANDED IN

Exercise 1

| Instruction | Memory | Register |
|----------------------------|--|------------------------------------|
| Example | \$005000 54 AF 00 40 E7 21 48 C0 | A0 = \$00005004 A1 = \$0000500C |
| Example | \$005008 C9 10 11 C8 D4 36 FF 88 | No change |
| MOVE.W 20482,(A0)+ | | |
| MOVE.B #28,16(A0,D1.W) | | |
| MOVE.L -6(A1),-18(A2,D2.W) | | |

Exercise 2

| Operation | Size (bits) | Result (hexadecimal) | N | Z | V | C |
|-------------------------|-------------|----------------------|---|---|---|---|
| \$72 + \$91 | 8 | | | | | |
| \$00000072 + \$FFFFFF91 | 32 | | | | | |

Exercise 3

| | | |
|--|----------------|----------------|
| Values of registers after the execution of the program. Use the 32-bit hexadecimal representation. | | |
| D1 = \$ | D2 = \$ | D3 = \$ |

Exercise 4

Final value of **D1** : **\$43215687** . **Be careful**, use **three** lines of instructions at the most.

Exercise 5

next_str

two_by_two_swap

swap_all