# Key to Midterm Exam S4
# Computer Architecture

**Duration: 1 hr 30 min**

**Write answers only on the answer sheet.**

## Exercise 1 (4 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 = $FFFF0015   A0 = $00005000   PC = $00006000
D1 = $12340004   A1 = $00005008
D2 = $FFFFFFE1   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 (3 points)

Complete the table shown on the answer sheet. Determine the missing number for each addition in order to match the given flags (use the hexadecimal representation). **If multiple answers are possible, choose the smallest one**.

## Exercise 3 (4 points)

Let us consider the following program. Complete the table shown on the answer sheet.

```
Main        move.l  #$85A51000,d7

next1       moveq.l #1,d1
            cmpi.w  #$80,d7
            blt     next2
            moveq.l #2,d1

next2       move.l  d7,d2
            ror.l   #4,d2
            swap    d2
            rol.w   #8,d2
            rol.b   #4,d2

next3       clr.l   d3
            move.l  d7,d0
loop3       addq.l  #1,d3
            subq.w  #2,d0
            bne     loop3

next4       clr.l   d4
            move.l  d7,d0
loop4       addq.l  #1,d4
            dbra    d0,loop4      ; DBRA = DBF
```

## Exercise 4 (9 points)

All 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**. **Be careful. All the subroutines must contain 15 lines of instructions at the most.**

Structure of a bitmap:

| **Field** | **Size (bits)** | **Encoding** | **Description** |
|---|---|---|---|
| WIDTH | 16 | Unsigned integer | Width of the bitmap in pixels |
| HEIGHT | 16 | Unsigned integer | Height of the bitmap in pixels |
| MATRIX | Variable | Bitmap | Dot matrix of the bitmap.<br>If a bit is 0, the displayed pixel is black.<br>If a bit is 1, the displayed pixel is white. |

Structure of a sprite:

| **Field** | **Size (bits)** | **Encoding** | **Description** |
|---|---|---|---|
| STATE | 16 | Unsigned integer | Current display state of the sprite<br>Only two possible values: HIDE = 0 or SHOW = 1 |
| X | 16 | Signed integer | Abscissa of the sprite |
| Y | 16 | Signed integer | Ordinate of the sprite |
| BITMAP1 | 32 | Unsigned integer | Address of the first bitmap |
| BITMAP2 | 32 | Unsigned integer | Address of the second bitmap |

We assume that the size of the bitmap 1 is always equal to that of the bitmap 2.

Constants that are already defined:

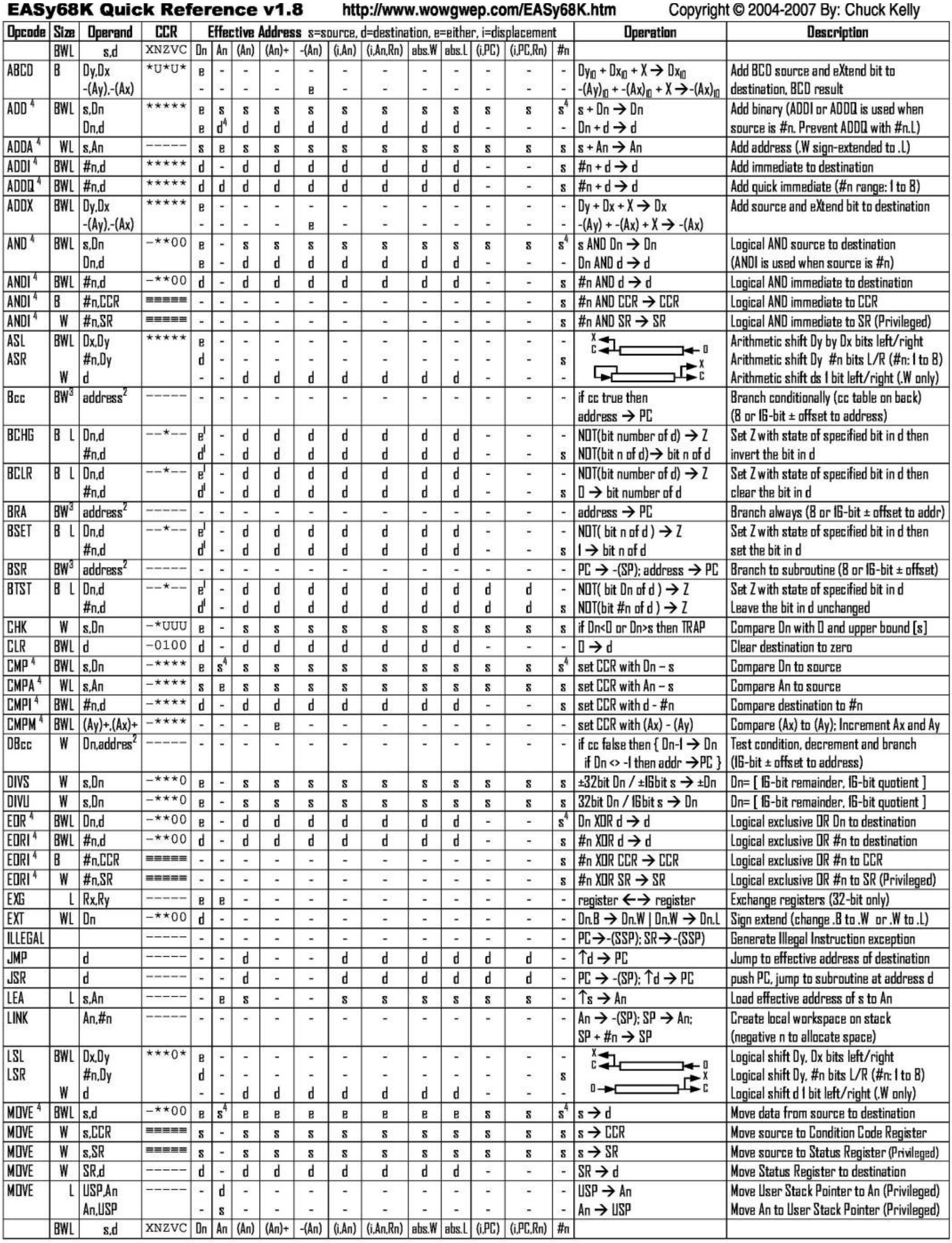
```
VIDEO_START         equ     $ffb500         ; Starting address of the video memory
VIDEO_SIZE          equ     (480*320/8)     ; Size in bytes of the video memory

WIDTH               equ     0
HEIGHT              equ     2
MATRIX              equ     4

STATE               equ     0
X                   equ     2
Y                   equ     4
BITMAP1             equ     6
BITMAP2             equ     10

HIDE                equ     0
SHOW                equ     1
```

1. Write the **FillScreen** subroutine that fills the video memory with a 32-bit integer.
   Input: **D0.L** = A 32-bit integer used to fill the video memory.

2. Write the **GetRectangle** subroutine that returns the coordinates of the rectangle that marks out the boundaries of a sprite.
   Input: **A0.L** = Address of the sprite.
   Outputs: **D1.W** = Abscissa of the top left corner of the sprite.
   **D2.W** = Ordinate of the top left corner of the sprite.
   **D3.W** = Abscissa of the bottom right corner of the sprite.
   **D4.W** = Ordinate of the bottom right corner of the sprite.

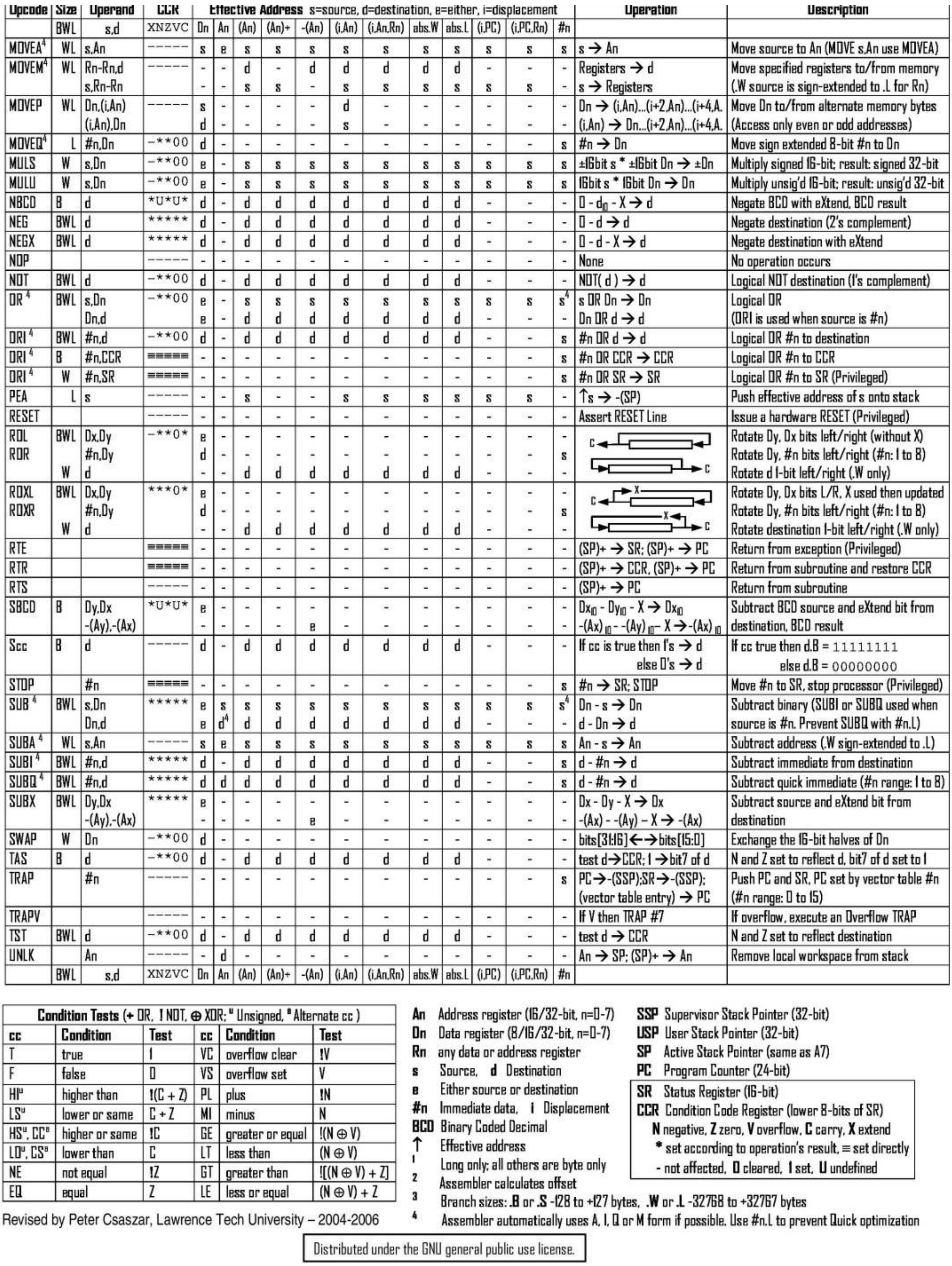
3. Write the **MoveSprite** subroutine that moves a sprite in a relative way. If the new position of the sprite is off the screen, the sprite must remain still (the new position will be ignored).
   Inputs: **A1.L** = Address of a sprite.
   **D1.W** = Relative horizontal displacement in pixels (16-bit signed integers).
   **D2.W** = Relative vertical displacement in pixels (16-bit signed integers).
   Outputs: **D0.L** returns *false* (0) if the sprite has not moved (its new position was out of the screen).
   **D0.L** returns *true* (1) if the sprite has moved.

   To know if a sprite is out of the screen, you can call the **IsOutOfScreen** subroutine. We will assume that this subroutine has already been written (you do not have to write it).
   Inputs: **A0.L** = Address of a bitmap.
   **D1.W** = Abscissa of the bitmap in pixels (16-bit signed integer).
   **D2.W** = Ordinate of the bitmap in pixels (16-bit signed integer).
   Outputs: **Z** returns *false* (0) if the bitmap is not out of the screen.
   **Z** returns *true* (1) if the bitmap is out of the screen.



**EASy68K Quick Reference v1.8** http://www.wowgwep.com/EASy68K.htm Copyright © 2004-2007 By: Chuck Kelly

| 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<br>-(Ay),-(Ax) | *U*U* | e<br>- | -<br>- | -<br>- | -<br>- | -<br>e | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | $Dy_{10} + Dx_{10} + X \rightarrow Dx_{10}$<br>$-(Ay)_{10} + -(Ax)_{10} + X \rightarrow -(Ax)_{10}$ | Add BCD source and eXtend bit to destination, BCD result |
| ADD [4] | BWL | s,Dn<br>Dn,d | ***** | e<br>e | s<br>d[4] | s<br>d | s<br>d | s<br>d | s<br>d | s<br>d | s<br>d | s<br>d | s<br>- | s<br>- | s[4]<br>- | s + Dn → Dn<br>Dn + d → 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 → An | Add address (.W sign-extended to .L) |
| ADDI [4] | BWL | #n,d | ***** | d | - | d | d | d | d | d | d | d | - | - | s | #n + d → d | Add immediate to destination |
| ADDQ [4] | BWL | #n,d | ***** | d | d | d | d | d | d | d | d | d | - | - | s | #n + d → d | Add quick immediate (#n range: 1 to 8) |
| ADDX | BWL | Dy,Dx<br>-(Ay),-(Ax) | ***** | e<br>- | -<br>- | -<br>- | -<br>- | -<br>e | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | Dy + Dx + X → Dx<br>-(Ay) + -(Ax) + X → -(Ax) | Add source and eXtend bit to destination |
| AND [4] | BWL | s,Dn<br>Dn,d | -**00 | e<br>e | -<br>- | s<br>d | s<br>d | s<br>d | s<br>d | s<br>d | s<br>d | s<br>d | s<br>- | s<br>- | s[4]<br>- | s AND Dn → Dn<br>Dn AND d → 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 | - | - | s | #n AND d → d | Logical AND immediate to destination |
| ANDI [4] | B | #n,CCR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n AND CCR → CCR | Logical AND immediate to CCR |
| ANDI [4] | W | #n,SR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n AND SR → SR | Logical AND immediate to SR (Privileged) |
| ASL<br>ASR | BWL<br><br>W | Dx,Dy<br>#n,Dy<br>d | ***** | e<br>d<br>- | -<br>-<br>- | -<br>-<br>d | -<br>-<br>d | -<br>-<br>d | -<br>-<br>d | -<br>-<br>d | -<br>-<br>d | -<br>-<br>d | -<br>-<br>- | -<br>-<br>- | -<br>s<br>- | | Arithmetic shift Dy by Dx bits left/right<br>Arithmetic shift Dy #n bits L/R (#n: 1 to 8)<br>Arithmetic shift ds 1 bit left/right (.W only) |
| Bcc | BW[3] | address[2] | ----- | - | - | - | - | - | - | - | - | - | - | - | - | if cc true then address → PC | Branch conditionally (cc table on back) (8 or 16-bit ± offset to address) |
| BCHG | B L | Dn,d<br>#n,d | --*-- | e[1]<br>d[1] | -<br>- | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | -<br>- | -<br>- | -<br>s | NOT(bit number of d) → Z<br>NOT(bit n of d)→ bit n of d | Set Z with state of specified bit in d then invert the bit in d |
| BCLR | B L | Dn,d<br>#n,d | --*-- | e[1]<br>d[1] | -<br>- | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | -<br>- | -<br>- | -<br>s | NOT(bit number of d) → Z<br>0 → 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 → PC | Branch always (8 or 16-bit ± offset to addr) |
| BSET | B L | Dn,d<br>#n,d | --*-- | e[1]<br>d[1] | -<br>- | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | -<br>- | -<br>- | -<br>s | NOT( bit n of d ) → Z<br>1 → bit n of d | Set Z with state of specified bit in d then set the bit in d |
| BSR | BW[3] | address[2] | ----- | - | - | - | - | - | - | - | - | - | - | - | - | PC → -(SP); address → PC | Branch to subroutine (8 or 16-bit ± offset) |
| BTST | B L | Dn,d<br>#n,d | --*-- | e[1]<br>d[1] | -<br>- | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | d<br>d | -<br>s | NOT( bit Dn of d ) → Z<br>NOT(bit #n of d ) → Z | Set Z with state of specified bit in d<br>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 → d | Clear destination to zero |
| CMP [4] | BWL | s,Dn | -**** | e | s[4] | s | s | s | s | s | s | s | s | s | s[4] | 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 | - | - | s | 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,addres[2] | ----- | - | - | - | - | - | - | - | - | - | - | - | - | if cc false then { Dn-1 → Dn<br>if Dn <> -1 then addr →PC } | Test condition, decrement and branch (16-bit ± offset to address) |
| DIVS | W | s,Dn | -***0 | e | - | s | s | s | s | s | s | s | s | s | s | ±32bit Dn / ±16bit s → ±Dn | Dn= [ 16-bit remainder, 16-bit quotient ] |
| DIVU | W | s,Dn | -***0 | e | - | s | s | s | s | s | s | s | s | s | s | 32bit Dn / 16bit s → Dn | Dn= [ 16-bit remainder, 16-bit quotient ] |
| EOR [4] | BWL | Dn,d | -**00 | e | - | d | d | d | d | d | d | d | - | - | s[4] | Dn XOR d → d | Logical exclusive OR Dn to destination |
| EORI [4] | BWL | #n,d | -**00 | d | - | d | d | d | d | d | d | d | - | - | s | #n XOR d → d | Logical exclusive OR #n to destination |
| EORI [4] | B | #n,CCR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n XOR CCR → CCR | Logical exclusive OR #n to CCR |
| EORI [4] | W | #n,SR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n XOR SR → SR | Logical exclusive OR #n to SR (Privileged) |
| EXG | L | Rx,Ry | ----- | e | e | - | - | - | - | - | - | - | - | - | - | register ←→ register | Exchange registers (32-bit only) |
| EXT | WL | Dn | -**00 | d | - | - | - | - | - | - | - | - | - | - | - | Dn.B → Dn.W \| Dn.W → Dn.L | Sign extend (change .B to .W or .W to .L) |
| ILLEGAL | | | ----- | - | - | - | - | - | - | - | - | - | - | - | - | PC→-(SSP); SR→-(SSP) | Generate Illegal Instruction exception |
| JMP | | d | ----- | - | - | d | - | - | d | d | d | d | d | d | - | ↑d → PC | Jump to effective address of destination |
| JSR | | d | ----- | - | - | d | - | - | d | d | d | d | d | d | - | PC → -(SP); ↑d → PC | push PC, jump to subroutine at address d |
| LEA | L | s,An | ----- | - | e | s | - | - | s | s | s | s | s | s | - | ↑s → An | Load effective address of s to An |
| LINK | | An,#n | ----- | - | - | - | - | - | - | - | - | - | - | - | - | An → -(SP); SP → An;<br>SP + #n → SP | Create local workspace on stack (negative n to allocate space) |
| LSL<br>LSR | BWL<br><br>W | Dx,Dy<br>#n,Dy<br>d | ***0* | e<br>d<br>- | -<br>-<br>- | -<br>-<br>d | -<br>-<br>d | -<br>-<br>d | -<br>-<br>d | -<br>-<br>d | -<br>-<br>d | -<br>-<br>d | -<br>-<br>- | -<br>-<br>- | -<br>s<br>- | | Logical shift Dy, Dx bits left/right<br>Logical shift Dy, #n bits L/R (#n: 1 to 8)<br>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[4] | s → d | Move data from source to destination |
| MOVE | W | s,CCR | ===== | s | - | s | s | s | s | s | s | s | s | s | s | s → CCR | Move source to Condition Code Register |
| MOVE | W | s,SR | ===== | s | - | s | s | s | s | s | s | s | s | s | s | s → SR | Move source to Status Register (Privileged) |
| MOVE | W | SR,d | ----- | d | - | d | d | d | d | d | d | d | - | - | - | SR → d | Move Status Register to destination |
| MOVE | L | USP,An<br>An,USP | ----- | -<br>- | d<br>s | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | -<br>- | USP → An<br>An → USP | Move User Stack Pointer to An (Privileged)<br>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 | | |



| 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 → An | Move source to An (MOVE s,An use MOVEA) |
| MOVEM[4] | WL | Rn-Rn,d | ----- | - | - | d | - | d | d | d | d | d | - | - | - | Registers → d | Move specified registers to/from memory |
| | | s,Rn-Rn | | - | - | s | s | - | s | s | s | s | s | s | - | s → Registers | (.W source is sign-extended to .L for Rn) |
| MOVEP | WL | Dn,(i,An) | ----- | s | - | - | - | - | d | - | - | - | - | - | - | Dn → (i,An)...(i+2,An)...(i+4,A. | Move Dn to/from alternate memory bytes |
| | | (i,An),Dn | | d | - | - | - | - | s | - | - | - | - | - | - | (i,An) → Dn...(i+2,An)...(i+4,A. | (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 | ±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 | 16bit s * 16bit Dn → Dn | Multiply unsig'd 16-bit; result: unsig'd 32-bit |
| NBCD | B | d | *U*U* | 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 | - | - | - | 0 - d → d | Negate destination (2's complement) |
| NEGX | BWL | d | ***** | d | - | d | d | d | d | d | d | d | - | - | - | 0 - d - X → d | Negate destination with eXtend |
| NOP | | | ----- | - | - | - | - | - | - | - | - | - | - | - | - | None | No operation occurs |
| NOT | BWL | d | -**00 | d | - | d | d | d | d | d | d | d | - | - | - | NOT( d ) → d | Logical NOT destination (1's complement) |
| OR[4] | BWL | s,Dn | -**00 | e | - | s | s | s | s | s | s | s | s | s | s[4] | s OR Dn → Dn | Logical OR |
| | | Dn,d | | e | - | d | d | d | d | d | d | d | - | - | - | Dn OR d → d | (ORI is used when source is #n) |
| ORI[4] | BWL | #n,d | -**00 | d | - | d | d | d | d | d | d | d | - | - | s | #n OR d → d | Logical OR #n to destination |
| ORI[4] | B | #n,CCR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n OR CCR → CCR | Logical OR #n to CCR |
| ORI[4] | W | #n,SR | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n OR SR → SR | Logical OR #n to SR (Privileged) |
| PEA | L | 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 | -**0* | e | - | - | - | - | - | - | - | - | - | - | - | C | Rotate Dy, Dx bits left/right (without X) |
| ROR | | #n,Dy | | d | - | - | - | - | - | - | - | - | - | - | s | | Rotate Dy, #n bits left/right (#n: 1 to 8) |
| | W | d | | - | - | d | d | d | d | d | d | d | - | - | - | C | Rotate d 1-bit left/right (.W only) |
| ROXL | BWL | Dx,Dy | ***0* | e | - | - | - | - | - | - | - | - | - | - | - | C X | Rotate Dy, Dx bits L/R, X used then updated |
| ROXR | | #n,Dy | | d | - | - | - | - | - | - | - | - | - | - | s | | Rotate Dy, #n bits left/right (#n: 1 to 8) |
| | W | d | | - | - | d | d | d | d | d | d | d | - | - | - | X C | Rotate destination 1-bit left/right (.W only) |
| 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 | *U*U* | e | - | - | - | - | - | - | - | - | - | - | - | $Dx_{10}$ - $Dy_{10}$ - X → $Dx_{10}$ | Subtract BCD source and eXtend bit from |
| | | -(Ay),-(Ax) | | - | - | - | - | e | - | - | - | - | - | - | - | $-(Ax)_{10}$ - $-(Ay)_{10}$ – X → $-(Ax)_{10}$ | destination, BCD result |
| Scc | B | d | ----- | d | - | d | d | d | d | d | d | d | - | - | - | If cc is true then 1's → d | If cc true then d.B = 11111111 |
| | | | | | | | | | | | | | | | | else 0's → d | else d.B = 00000000 |
| STOP | | #n | ===== | - | - | - | - | - | - | - | - | - | - | - | s | #n → SR; STOP | Move #n to SR, stop processor (Privileged) |
| SUB[4] | BWL | s,Dn | ***** | e | s | s | s | s | s | s | s | s | s | s | s[4] | Dn - s → Dn | Subtract binary (SUBI or SUBQ used when |
| | | Dn,d | | e | d[4] | d | d | d | d | d | d | d | - | - | - | d - Dn → d | 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 | An - s → An | Subtract address (.W sign-extended to .L) |
| SUBI[4] | BWL | #n,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 | - | - | s | d - #n → d | Subtract quick immediate (#n range: 1 to 8) |
| SUBX | BWL | Dy,Dx | ***** | e | - | - | - | - | - | - | - | - | - | - | - | Dx - Dy - X → Dx | Subtract source and eXtend bit from |
| | | -(Ay),-(Ax) | | - | - | - | - | e | - | - | - | - | - | - | - | -(Ax) - -(Ay) – X → -(Ax) | 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 | - | - | - | 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); | Push PC and SR, PC set by vector table #n |
| | | | | | | | | | | | | | | | | (vector table entry) → PC | (#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 | - | - | - | test d → CCR | N and Z set to reflect destination |
| UNLK | | 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; [u] Unsigned, [a] Alternate cc ) | | | | | |
|---|---|---|---|---|---|
| cc | Condition | Test | cc | Condition | Test |
| T | true | 1 | VC | overflow clear | !V |
| F | false | 0 | VS | overflow set | V |
| HI[u] | higher than | !(C + Z) | PL | plus | !N |
| LS[u] | lower or same | C + Z | MI | minus | N |
| HS[u], CC[a] | higher or same | !C | GE | greater or equal | !(N ⊕ V) |
| LO[u], CS[a] | 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 |

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

**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

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`<br>A1 = `$0000500C` |
| Example | `$005008  C9 10 11 C8 D4 36` **`FF`** `88` | No change |
| `MOVE.L #4507,-(A1)` | `$005000  54 AF 18 B9` **`00 00 11 9B`** | A1 = `$00005004` |
| `MOVE.B $5009,-6(A1)` | `$005000  54 AF` **`10`** `B9 E7 21 48 C0` | No change |
| `MOVE.W 8(A1),-37(A2,D0.W)` | `$005000` **`13 79`** `18 B9 E7 21 48 C0` | No change |
| `MOVE.L -4(A2),$21(A0,D2.L)` | `$005000  54 AF` **`D4 36 1F 88`** `48 C0` | No change |

## Exercise 2

| Operation | Size (bits) | Missing Number (hexadecimal) | N | Z | V | C |
|---|---|---|---|---|---|---|
| `$80 + $?` | 8 | **`$00`** | 1 | 0 | 0 | 0 |
| `$8000 + $?` | 16 | **`$8000`** | 0 | 1 | 1 | 1 |
| `$80000000 + $?` | 32 | **`$80000001`** | 0 | 0 | 1 | 1 |

## Exercise 3

Values of registers after the execution of the program.

**Use the 32-bit hexadecimal representation.**

| | |
|---|---|
| **D1** = $00000002 | **D3** = $00000800 |
| **D2** = $51005A80 | **D4** = $00001001 |

## Exercise 4

```
FillScreen          movem.l d7/a0,-(a7)

                    lea     VIDEO_START,a0
                    move.w  #VIDEO_SIZE/4-1,d7

\loop               move.l  d0,(a0)+
                    dbra    d7,\loop

                    movem.l (a7)+,d7/a0
                    rts
```

```
GetRectangle        move.l  a0,-(a7)

                    move.w  X(a0),d1
                    move.w  Y(a0),d2

                    movea.l BITMAP1(a0),a0

                    move.w  WIDTH(a0),d3
                    add.w   d1,d3
                    subq.w  #1,d3

                    move.w  HEIGHT(a0),d4
                    add.w   d2,d4
                    subq.w  #1,d4

                    movea.l (a7)+,a0
                    rts
```

```
MoveSprite          movem.l d1/d2/a0,-(a7)

                    add.w   X(a1),d1
                    add.w   Y(a1),d2

                    movea.l BITMAP1(a1),a0

                    jsr     IsOutOfScreen
                    beq     \false

                    move.w  d1,X(a1)
                    move.w  d2,Y(a1)

\true               moveq.l #1,d0
                    bra     \quit

\false              moveq.l #0,d0
\quit               movem.l (a7)+,d1/d2/a0
                    rts
```