Trajan - Assembler Description

Bogdan Pasca

May 19, 2009

1 Getting the assembler

The assembler, together with the simulator of the Trajan processor are distributed in the package that can be downloaded from

`http://perso.ens-lyon.fr/bogdan.pasca/teaching.html`

2 Using the assembler

In order to compile the **.asm** in an **.obj**:

```
./asm < input.asm > output.obj
```

where *input.asm* represents the assembler input file and the *output.obj* represents the output file containing the object code for the simulator.

3 Instruction description

- `ldl imm`

  Loads the low part (LSB 8-bits) of the Accumulator (reg 0) with the immediate value while clearing the high part (MSB 8-bits) of the accumulator. Immediate expected as unsigned decimal integer (8-bit) or hexadecimal.

  Example: `ldl 16` , `ldl 0x10`

- `ldh imm`

  Loads the high part (LSB 8-bits) of the Accumulator (reg 0) with the immediate value while leaving unchanged the low part (LSB 8-bits) of the accumulator. Immediate expected as unsigned decimal integer (8-bit) or hexadecimal.

  Example: `ldh 16` , `ldh 0x10`

- `mv2reg regX`

  Moves the contents of the accumulator in the register X, where X $\in$ 0..127

  Example: `mv2reg r16`

- `mv2acc regX`

  Moves the contents of the the register X, where X $\in$ 0..127, in the accumulator.

  Example: `mv2acc r2`

- `rd regX`

  Reads the contents of the memory at the address present in the accumulator and copies it into register X.

  Example: `rd r2 -> r2=M[Acc]`

- `wr regX`

  Writes the contents of register X in the memory at the address present in the accumulator.

  Example: `wr r2 -> M[Acc]=r2`

- `rdinc regX`

  Reads the contents of the memory at the address present in the accumulator and copies it into register X and then increments the accumulator.

  Example: `rdinc r2 -> r2=M[Acc], Acc=Acc+1`

- `wrinc regX`

  Writes the contents of register X in the memory at the address present in the accumulator and then increments the accumulator.

  Example: `wrinc r2 -> M[Acc]=r2, Acc=Acc+1`

- `rddec regX`

  Reads the contents of the memory at the address present in the accumulator and copies it into register X and then decrements the accumulator.

  Example: `rddec r2 -> r2=M[Acc], Acc=Acc-1`

- `wrdec regX`

  Writes the contents of register X in the memory at the address present in the accumulator and then decrements the accumulator.

  Example: `wrdec r2 -> M[Acc]=r2, Acc=Acc-1`

- `jmp regX`

  Absolute jump. Copies the value of register X into the program counter (PC)

  Example: `jmp r2 -> PC=r2`

- `jmr regX`

  Long relative jump. Adds to the ProgramCounter the signed 2's complement value of register X.

  Example: `jmr r2 -> PC=PC+signed(r2)`

- `jmi imm/label`

  Absolute immediate jump. Copies the immediate value into the program counter (PC). Immediate expected as unsigned decimal integer (8-bit) or hexadecimal. Immediate can be replaced by a label with the condition that the address of the label is addressable on unsigned 8-bits.

  Example: `jmi 45` , `jmi 0x11`, `jmi labelName -> PC=imm`

- `jmri imm/label`

  Relative immediate jump. Adds the signed immediate value to the program counter (PC). Immediate expected as unsigned decimal integer (8-bit) or hexadecimal. Immediate can be replaced by a label with the condition that the address of the label is addressable on 2's complement 8-bits starting from PC.

  Example: `jmri 45` , `jmri 0x11`, `jmi labelName > PC=PC+signed(imm)`

- `jsr regX`

  Jump to far subroutine. First the program counter is copied at the address of the stack pointer (register 127). Second, the stack pointer is incremented and finally, the program counter (PC) is assigned the value of register X.

  Example: `jsr reg3` , `-> Mem[reg[127]]=PC, reg[127]=reg[127]+1, PC=regX`

- `jsri imm/label`

  Jump to close subroutine. First the program counter is copied at the address of the stack pointer (register 127). Second, the stack pointer is incremented and finally, the program counter (PC) incremented with the signed value of the immediate.

  Example: `jsri 45` , `jsri my_subroutine -> Mem[reg[127]]=PC, reg[127]=reg[127]+1, PC=PC+signed(imm)`

- `rts`

  Return from subroutine. First the stack pointer (stored in register 127) is decremented. Second, the contents of the memory at this address value are copied in the PC.

  Example: `rts` , `reg[127]=reg[127]-1, PC=Mem[reg[127]]]`

- `nop`

  No operation.

  Example: `nop` , `PC=PC+1`

- `add regX`

  Addition. Adds the 2's complement representations of the accumulator and register X. It may set the flags Zero(Z), Overflow(O) and Negative(N) flags.

  Z is set when all bits of the result are 0.

  N when the result is negative (MSB=1)

O when the result cannot be represented in 2's complement on 16 bits

Example: `add reg5 , Acc=signed(Acc)+signed(reg5)`

- `sub regX`

  Subtraction. Subtracts the 2's complement representations of register X from the accumulator. It may set the flags Zero(Z), Overflow(O) and Negative(N) flags.

  Z is set when all bits of the result are 0.

  N when the result is negative (MSB=1)

  O when the result cannot be represented in 2's complement on 16 bits

  Example: `sub reg5 , Acc=signed(Acc)-signed(reg5)`

- `mul regX`

  Multiplication. The operation directly considers the 2's complement low 8-bits of both register X and the accumulator. The multiplication is performed on these values. The result is a 2's complement number. It may set the flags Zero(Z) and Negative(N) flags. It cannot overflow so when *set flags* is activated Overflow(O)=0..

  Z is set when all bits of the result are 0.

  N when the result is negative (MSB=1)

  Example: `mul reg5 , Acc= signed(low(Acc))*signed(low(reg5))`

- `cmp regX`

  Comparison. Subtracts the 2's complement representations of register X from the accumulator. but does not write back the result. It may set the flags Zero(Z), Overflow(O) and Negative(N) flags.

  Z is set when all bits of the result are 0.

  N when the result is negative (MSB=1)

  O when the result cannot be represented in 2's complement on 16 bits

  Example: `cmp reg5 , -No effect except flags`

- `swap regX`

  Swap. Swaps the high and the low parts of register X. It may set the flags Zero(Z) and Negative(N) flags. When *set flags* is activated Overflow(O)=0.

  Z is set when all bits of the result are 0.

  N when the result is negative (MSB=1)

  Example: `swap reg5 , reg5=low(reg5)&high(reg5)`

- `clr regX`

  Clear. Clears the content of register X. It may set the flags Zero(Z)=1 and Negative(N)=0 and Overflow(O)=0 flags.

  Example: `clr reg5` , `reg5=0`

- `and regX`

  AND. Bitwise and between register X and the Accumulator. It may set the flags Zero(Z) and Negative(N). If *set flags* is activated Overflow(O)=0.

  Example: `and reg5` , `acc=(acc and reg5)`

- `or regX`

  OR. Bitwise or between register X and the Accumulator. It may set the flags Zero(Z) and Negative(N). If *set flags* is activated Overflow(O)=0.

  Example: `or reg5` , `acc=(acc or reg5)`

- `xor regX`

  XOR. Bitwise xor between register X and the Accumulator. It may set the flags Zero(Z) and Negative(N). If *set flags* is activated Overflow(O)=0.

  Example: `xor reg5` , `acc=(acc xor reg5)`

- `not regX`

  NOT. The negated value of register X is copied into the accumulator. It may set the flags Zero(Z) and Negative(N). If *set flags* is activated Overflow(O)=0.

  Example: `not reg5` , `acc=not(reg5)`

- `lsr regX`

  Logical shift Right. Shifts the contents of the accumulator to the right a number of positions equal to the value stored in regX.

  It may set the flags Zero(Z) and Negative(N). If *set flags* is activated Overflow(O)=0.

  Example: `lsr reg5` , `acc=acc >> reg5`

- `lsl regX`

  Logical shift Left. Shifts the contents of the accumulator to the left a number of positions equal to the value stored in regX.

  It may set the flags Zero(Z) and Negative(N). If *set flags* is activated Overflow(O)=0.

  Example: `lsl reg5` , `acc=acc << reg5`

- `ror regX`

  Rotation Right. Rotates the contents of the accumulator to the right a number of positions equal to the value stored in regX. The bits lost at the LSB by rotating are inserted at MSB.

  It may set the flags Zero(Z) and Negative(N). If *set flags* is activated Overflow(O)=0.

  Example: `ror reg5` , `acc= acc rotate right reg5`

- `rol regX`

  Rotation Left. Rotates the contents of the accumulator to the left a number of positions equal to the value stored in regX. The bits lost at the MSB by rotating are inserted at LSB.

  It may set the flags Zero(Z) and Negative(N). If *set flags* is activated Overflow(O)=0.

  Example: `rol reg5` , `acc=acc rotate left reg5`

- `block X imm`

  Block instruction. Reserves in memory a block of imm 16-bit words for the variable with the name X.

  Example: `block my_block 5`

- `integer Y`

  Integer declaration. Reserves in memory a 16-bit word for the integer variable denoted by Y.

  Example: `integer my_integer`

- `define C imm16`

  Constant definition. Allocates in the code memory a 16 bit word for the constant C which is initialized with the value of imm16 (a 16 bit immediate).

  Example: `define Prime 17`

- `ldvar varName regX`

  Load variable into register. Loads the value of the variable denoted by varName into the register regX.

  Example: `ldvar my_integer regX`

- `loadl [varName/label]'[high|low]`

  Load the low part of the Accumulator. Loads the low part of the accumulator with the high part or the low part of the address of a variable or a label.

  Example: `loadl my_label'low` , `loadl my_label'high` ,

  `loadl my_var'low` , `loadl my_var'high`

- `loadh [varName/label]'[high|low]`

  Load the high part of the Accumulator. Loads the low part of the accumulator with the high part or the low part of the address of a variable or a label.

  Example: `loadh my_label'low` , `loadh my_label'high` ,

  `loadh my_var'low` , `loadh my_var'high`

4 An example - deco.asm

```
init:
        ;the character to be printed is =
        ldl 61
        mv2reg r5

        ldl 0x00
        ldh 0x20
        mv2reg r3

        ldl 20
        mv2reg r4
        ldl 80

        mul r4
        add r4
        add r3

        ;this contains the address
        mv2reg r2

        ;number of characters
        ldl 20
        mv2reg r7

        clr r11

        ;two stripes of ==========
        ldl 2
        mv2reg r19

draw:
        ;gets the number of characters from r7
        ;gets the memory address where to start from r2
        ;zero is available in r11

        ;print one character

        mv2acc r2
        wrinc r5
        mv2reg r2

        ;decrement the number of characters

        ldl 1
        mv2reg r12
        mv2acc r7
        sub r12
```

```
        mv2reg r7
        ;

        ;compare to zero
        !s cmp r11
?p NE jmri draw

        ldl 80
        mv2reg r20
        mv2acc r2
        add r20
        add r20
        sub r4
        mv2reg r2

        ldl 20
        mv2reg r7

        ldl 1
        mv2reg r12
        mv2acc r19
        sub r12
        mv2reg r19

        !s cmp r11
?p NE jmri draw

        ;print text <asr1 2008>
        ldl 6
        mv2reg r77
        ;set text
        clr r21

        ;a
        ldl 97
        mv2reg r30
        ;s
        ldl 115
        mv2reg r31
        ;r
        ldl 114
        mv2reg r32
        ;" "
        clr     r33
        ;2
        ldl 50
        mv2reg r34
        ;0
        ldl 48
        mv2reg r35
```

```
        ;9
        ldl 57
        mv2reg r36

        ; set memory address
        ldl 80
        mv2reg r55

        mv2acc r2
        add r77
        sub r55
        sub r55
        sub r55

        ;actually write
        wrinc r30
        wrinc r31
        wrinc r32
        wrinc r33
        wrinc r34
        wrinc r35
        wrinc r35
        wrinc r36

        ldl -3
end:
        loadl end'low
        loadh end'high
        jmp r0
```

5 A second example which loops forever

```
 begin:
    ; ldl example; fist decimal immediate, second binary immediate
    ldl 16
    ldl 0x10

        loadl enda'low
        loadh enda'high
        jmp r0
        jmri fin
enda:
        jmri begin
fin:
```