UoS educational processor

Characteristics:

- 8-bit, 4 register processor, Von Neumann architecture
- 3 clock cycle per instruction
- 16-bit instruction set (inspired by x86 ISA) with direct, indirect, immediate, register addressing
- Customizable instructions
- External memory bus (for program/data)
- I/O interface (as in microcontrollers)
- Implemented in VHDL
- Synthesizable

Registers and flags

```
There are 4 registers. They are encoded as follows in the
instruction:
RA:  00
RB:  01
RC:  10
RD:  11

There are 4 flags, that are set after the "cmp dst, src"
instruction.
     ZF: Zero flag
     SF: Sign flag
     CF: Carry flag
     OF: Overflow flag (not implemented/used yet)

Flags are used to store the result of a comparison operation.
The conditional jump operations check the status of the flag
to determine whether to jump or not.
```

Generalities

```
The CPU supports move, ALU, jump and external interface
instructions.

The general format of move and ALU instructions are of the
format:
     instr dst,src

instr is the name of the instruction (e.g. mov, add, sub). All
ALU instructions will take the data of dst, apply the
operation specified with the value of dst and src, and store
the result in dst.

dst is the destination where the result is placed. It is
always a register (dst is one of ra,rb,rc,rd) for an ALU
instruction. For move instructions the destination can be a
register or a memory location contained in a register. The
```

syntax for a mov to a memory location is to put the register in bracket (e.g. [ra]).
src is the source of the data. It is either a register or an immediate. An immediate is a value encoded in the instruction. With move instruction, the source can be directly the register value or the immediate (direct mode) or it can be fetched from memory (indirect mode) at the location indicated by the register or the immediate.

General instruction format

All instructions are 16 bit and follow this format:

instr(15..13)	instruction(12..8)	instruction(7..0)
Opcode	depends on the instruction	src

The opcode (operation code) indicates the type of operation corresponding to this instruction. The meaning of the remaining fields depend on this opcode.
src contains the source of data for the instruction (or is sometimes unused).
src can be an "immediate" (8-bit of data) that is stored in the instruction or a register, in which case only the 2 least significant bits are used to encode the register.

Move instruction (opcode 000)

Format:

```
mov dst, src
```

Instructions	instruction(15..8)								Instruction(7..0)							
Move	Opcode			$\overline{R}/I$	dd#m	sd#m	dreg		src (i or r*m*)							
mov r*n*, r*m*	0	0	0	0	0	0	r	r	-	-	-	-	-	-	r	r
mov r*n*, i	0	0	0	1	0	0	r	r	i	i	i	i	i	i	i	i
mov r*n*, [r*m*]	0	0	0	0	0	1	r	r	-	-	-	-	-	-	r	r
mov r*n*, [i]	0	0	0	1	0	1	r	r	i	i	i	i	i	i	i	i
mov [r*n*], r*m*	0	0	0	0	1	0	r	r	-	-	-	-	-	-	r	r
mov [r*n*], i	0	0	0	1	1	0	r	r	i	i	i	i	i	i	i	i

The destination can be a register or a memory location. The memory location is indicated by putting the name of the register in bracket.

The source can be a register, an immediate, or a memory location. If it is a memory location the address of the memory location can be provided in a register or in an immediate. Brackets are used to indicate that the source is a memory location.

dreg indicates the register used in the destination assignment. dd#m indicated whether the destination register is used as the target of the move, (dd#m=0) or provides the memory location for the move (dd#m=1).

The source src is an immediate providing the data or memory location of the source when $\overline{R}/I$=1 or a register providing the data or the memory location of the source when $\overline{R}/I$=0.

The source is read from the memory location indicated by src when sd#m=1, otherwise it is the data specified in src (direct or immediate according to $\overline{R}/I$).

Examples:

```
     mov ra, rb
Moves the content of register rb into ra
     mov ra, 35h
Moves the immediate 35h into ra
     mov rd, [ra]
Moves the data at memory location ra into register rd
     mov rd, [12h]
Moves the data at memory location 12h into register rd
     mov [rb], rc
Moves the content of register rc to the memory location rb.
     mov [rb], 07h
Moves the immediate 07h to the memory location rb.
```

ALU operations (opcode 001, 010, 011)

Format:

```
     instruction dst, src              (two operands)
     instruction dst                   (one operand)
```

ALU operations include two and one operand arithmetic/logic operations.

The destination dst is a register.

The source src can be a register or an immediate. The source is an immediate when $\overline{R}/I$=1 or a register when $\overline{R}/I$=0.

The instructions take dst as the first operand, src as the second operand, and place the result in dst. I.e.:

```
     dst <= instruction(dst,src)
```

See examples.

UoS educational processor - instruction set details

Two operands logic/arithmetic

Instructions	instruction(15..8)								Instruction(7..0)							
ALU 2 op	opcode			$\overline{R}/I$	ALU op		dreg		src (i or r*m*)							
add r*n*, r*m*	0	0	1	0	0	0	r	r	-	-	-	-	-	-	r	r
add r*n*, i	0	0	1	1	0	0	r	r	i	i	i	i	i	i	i	i
sub r*n*, r*m*	0	0	1	0	0	1	r	r	-	-	-	-	-	-	r	r
sub r*n*, i	0	0	1	1	0	1	r	r	i	i	i	i	i	i	i	i
and r*n*, r*m*	0	0	1	0	1	0	r	r	-	-	-	-	-	-	r	r
and r*n*, i	0	0	1	1	1	0	r	r	i	i	i	i	i	i	i	i
or r*n*, r*m*	0	0	1	0	1	1	r	r	-	-	-	-	-	-	r	r
or r*n*, i	0	0	1	1	1	1	r	r	i	i	i	i	i	i	i	i
xor r*n*, r*m*	0	1	0	0	0	0	r	r	-	-	-	-	-	-	r	r
xor r*n*, i	0	1	0	1	0	0	r	r	i	i	i	i	i	i	i	i

Two operands comparison

Test	opcode			$\overline{R}/I$	ALU op		dreg		immedite or r*m*							
cmp r*n*, r*m*	0	1	0	0	0	1	r	r	-	-	-	-	-	-	r	r
cmp r*n*, i	0	1	0	1	0	1	r	r	i	i	i	i	i	i	i	i

The comparison is realized by performing a subtraction of dst-src (without storing the result to the destination register).
The result of the comparison is stored in "flags"
Therefore:

dst=src: Zero flag set, Carry flag clear
dst>src: Zero flag clear, Carry flag clear
dst<src: Zero flag clear, Carry flag set

The conditional jump instructions check the flag bits to decide whether to perform the jump operation.

One operand logic/arithmetic

Instructions	instruction(15..8)								Instruction(7..0)							
ALU 1 op	opcode			ALU op			dreg									
not r	0	1	1	0	0	0	r	r	-	-	-	-	-	-	-	-
shr r	0	1	1	0	0	1	r	r	-	-	-	-	-	-	-	-
ror r	0	1	1	0	1	0	r	r	-	-	-	-	-	-	-	-
asr r	0	1	1	0	1	1	r	r	-	-	-	-	-	-	-	-
rol r	0	1	1	1	0	0	r	r	-	-	-	-	-	-	-	-

Examples:

```
     add  ra, rb
Stores ra+rb in rb
     sub  ra,3
```

```
stores ra-3 in ra.
     and  rd,55
stores the logical AND of rd and 55h in rd
     ror  rb
rotates right rb and store the result in rb.
```

Jumps (opcode 101)

```
Format:
     jxxx      src
```

Perform a conditional or unconditional jump (change the content of the instruction pointer) to a new memory location. The memory location can be an immediate, or the content of a register.
The unconditional jump instruction is jmp.

The conditional jump instructions are: je/jz, jne/jnz.

```
jz/je:          jump if zero/equal (Zero set)
jne/njz:        jump if not zero/equal (Zero clear)
ja:             jump if above (Zero clear, carry clear)
jb:             jump if below (Zero clear, carry set)
```

Instructions	instruction(15..8)								Instruction(7..0)							
Jump Unsigned: JA, JAE, JB, JBE	opcode			$\overline{R}/I$	Jump type				immedite / reg							
jmp r*n*	1	0	1	0	0	0	0	0	-	-	-	-	-	-	r	r
jmp i	1	0	1	1	0	0	0	0	i	i	i	i	i	i	i	i
je/jz r*n*	1	0	1	0	0	0	0	1	-	-	-	-	-	-	r	r
je/jz i	1	0	1	1	0	0	0	1	i	i	i	i	i	i	i	i
jne/jnz r*n*	1	0	1	0	1	0	0	1	-	-	-	-	-	-	r	r
jne/jnz i	1	0	1	1	1	0	0	1	i	i	i	i	i	i	i	i
ja r*n*	1	0	1	0	0	0	1	0	-	-	-	-	-	-	r	r
ja i	1	0	1	1	0	0	1	0	i	i	i	i	i	i	i	i
~~*jae rn*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*0*~~	~~*0*~~	~~*0*~~	~~*1*~~	~~*1*~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~*r*~~	~~*r*~~
~~*jae i*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*1*~~	~~*0*~~	~~*0*~~	~~*1*~~	~~*1*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~
~~*jbe rn*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*0*~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~*r*~~	~~*r*~~
~~*jbe i*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*1*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*0*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~
jb r*n*	1	0	1	0	1	0	1	1	-	-	-	-	-	-	r	r
jb i	1	0	1	1	1	0	1	1	i	i	i	i	i	i	i	i
~~*jc rn*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*0*~~	~~*0*~~	~~*1*~~	~~*0*~~	~~*0*~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~*r*~~	~~*r*~~
~~*jc i*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*0*~~	~~*0*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~
~~*jnc rn*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*1*~~	~~*0*~~	~~*0*~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~-~~	~~*r*~~	~~*r*~~
~~*jnc i*~~	~~*1*~~	~~*0*~~	~~*1*~~	~~*1*~~	~~*1*~~	~~*1*~~	~~*0*~~	~~*0*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~	~~*i*~~

~~Not implemented~~

UoS educational processor - instruction set details

External interface (opcode 110)

Syntax:

```
     out  src
     in   dst
```

Reads or write data from the external processor interface

Instructions	instruction(15..8)								Instruction(7..0)							
IO	opcode			$\overline{R}/I$	IO type		dreg		src							
out r*n*	1	1	0	0	0	0	-	-	-	-	-	-	-	-	r	r
out i	1	1	0	1	0	0	-	-	i	i	i	i	i	i	i	i
in r*n*	1	1	0	-	0	1	r	r	-	-	-	-	-	-	-	-