HERA: ***H**opefully **E**lectronics **R**unning **A**utomatically*

A slightly less simple CPU than the SPEAR. It is way more powerful due to:

- Significantly more control lines, thus more instructions
- An extra `C` register
- Direct read and write access to all registers
- Jump and branches as individual instructions (SPEAR solution was creative, but tedious)
- A 16-bit bus (i.e. jump in single cycle)
- 16-bit registers and ALU
- RAM size increased from 256 bytes to 128 KiB
- Additional NOR operation
- Multiple flags (carry out, zero, negative) in a proper status register
- Far better input/output capabilities
- Instructions without literal value fit into one byte

Features

- Full 16-bit architecture
  - 16-bit addressing for ROM and RAM (ROM: 64 KiB, RAM: 128 KiB)
    - Two RAM chips for direct 16-bit access
  - 16-bit bus, registers and ALU
- Usable digital I/O
  - 256 input ports (16-bit)
  - 256 output ports (16-bit), can be extended at expense of RAM size

Technical details

Instructions

The upper 4 bits of any instruction signify control lines for writing to the internal bus (everything but `STAT` uses 16-bit values):

Upper nibble	Name	Description
0	`LIT`	Literal value
1	`A`	A register
2	`B`	B register
3	`C`	C register
4	`RAM`	RAM
5	`RAM_P`	RAM pointer register
6	`PC`	Program counter register
7	`STAT`	Status register (8-bit)

Upper nibble	Name	Description
8	-	-
9	-	-
A	`ADD`	Adder (`A` + `B`)
B	`COM`	Two's complement (-`A`)
C	`NOR`	NOR gate ~(`A` | `B`)
D	-	-
E	-	-
F	-	-

If a literal is selected to be written to the bus (upper nibble is 0x0), it is read from the two bytes after the instruction in big-endian format.

The lower 4 bits signify control lines for reading from the internal bus:

Lower nibble	Name	Description
0	`VOID`	Ignore value
1	`A`	A register
2	`B`	B register
3	`C`	C register
4	`RAM`	RAM
5	`RAM_P`	RAM pointer register
6	`PC`	Program counter register
7	`STAT`	Status register (8-bit)
8	-	-
9	-	-
A	`A B`	A & B registers
B	`B RAM_P`	B & RAM_P registers
C	`C PC`	C & PC registers
D	`PC_C`	Program counter if carry flag set
E	`PC_Z`	Program counter if zero flag set
F	`PC_N`	Program counter if negative flag set

Assembly code Examples

Machine code Examples

Copying the value of the `A` register into the `B` register is done by instruction `0x12` (upper nibble 1 -> write `A` to bus, lower nibble 2 -> read from bus into `B`).

Setting `B` to a literal value is done by instruction `0x02` followed by two bytes in big-endian order (i.e. `0x02 0x12 0x34` -> `B=0x1234`).

Some larger programs showcasing some optimized instructions (all instructions in hexadecimal):

```
# This program writes a lookup-table (array) for showing a 4-bit value as a
hexadecimal character on a seven segment display using the following segments
(a=LSB, h=MSB):

#  aaa
# f   b
# f   b
# f   b
#  ggg
# e   c
# e   c
# e   c
#  ddd  h

# Using lower nibble 0xB can be a very efficient way to count a pointer (by
setting the operand and the pointer in a single instruction), the counting offset
(or "step") is determined by the A register.

(00) 01 00 01   # A = 0x01 (increment pointer while writing lookup-table)
(03) 0b 00 80   # B = RAM_P = 0x0080 (base pointer of lookup-table)
(06) 04 00 3f   # Write 0x3f (character '0') at address 0x0080
(09) ab         # Add into B register and RAM_P (increment pointer by 1)
(0A) 04 00 06   # Write 0x06 (character '1') at address 0x0081
(0D) ab         #
(0E) 04 00 5b   # '2'
(11) ab         #
(12) 04 00 4f   # '3'
(15) ab         #
(16) 04 00 66   # '4'
(19) ab         #
(1A) 04 00 6d   # '5'
(1D) ab         #
(1E) 04 00 7d   # '6'
(21) ab         #
(22) 04 00 07   # '7'
(25) ab         #
(26) 04 00 7f   # '8'
(29) ab         #
(2A) 04 00 6f   # '9'
(2D) ab         #
(2E) 04 00 77   # 'A'
(31) ab         #
```

```
(32) 04 00 7c   # 'b'
(35) ab         #
(36) 04 00 39   # 'C'
(39) ab         #
(3A) 04 00 5e   # 'd'
(3D) ab         #
(3E) 04 00 79   # 'E'
(41) ab         #
(42) 04 00 71   # 'F'

# The following instructions will load the input register and perform a binary AND
with 0x000f to get only the lower 4 bits (AND is performed by inverting both
inputs to a NOR operation). The result will be added to the base pointer of the
lookup-table (0x0080) and the output register will be set to the value at that
address to perform the lookup.

(45) 05 01 00   # RAM_P to input register 0x0100
(48) 4a         # Load input into A and B registers to prepare binary NOT
(49) c1         # NOR into A register (NOT of input)
(4A) 02 ff f0   # B = 0xfff0 (NOT of bitmask for lower 4 bits)
(4D) c1         # NOR into A register (AND of input and bitmask)
(4E) 02 00 80   # B = 0x0080 (base pointer of lookup-table)
(51) a5         # Add into RAM_P (pointer to element in lookup-table)
(52) 41         # Load value from lookup-table into A
(53) 05 02 00   # RAM_P to output register 0x0200
(56) 14         # Output A
(57) 06 00 45   # Loop lookup to continuously update (PC = 0x0045)
```