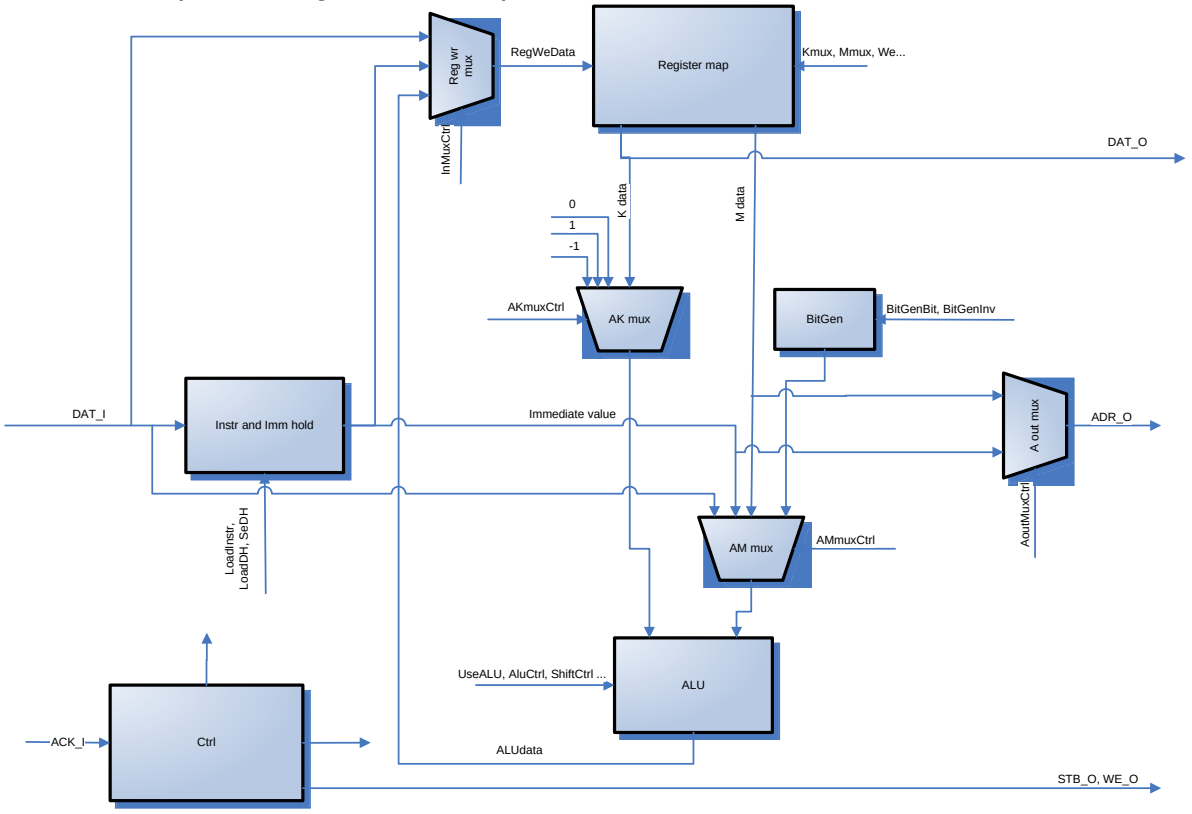
Short description of Alwcpu66

Alwcpu66 is ”**A L**ight **W**eight **CPU**” with 1**6** address and 1**6** databits.
The cpu is designed with the following in mind:

- low logic footprint in FPGA’s
- possibility to skip parts of cpu that is not needed to save logic resources

Architecture

Here is a simplified diagram of the cpu core.



Registers

Registers:

Register	Name	Function
0	PC	Program counter (16 bits)
1	SP	Stack pointer (16 bits)
2	ST	Status
3	MAP	Register map adjustment
7-4	GP7-4	General purpose registers 7-4 (16 bits)
15-8	GP15-8	General purpose registers 15-8 (16 bits)

Registers 3-0 are cleared on reset of processor, register 16-4 are not cleared!

Status bits:

Bit	Name	Function	Comment
0	N	Neg	Set if result was negative
1	Z	Zero	Set if result was zero

2	C	Carry (Borrow)	Set if a carry/borrow occurred in last operand
3	V	V, Signed overflow	Set if a signed overflow occurred
7-4		(reserved)	Always read as ‘0’
8	LT	LessThen	(N xor V)
9	LE	Lessthen or Equal	(Z or (N xor V))
10	LS	Less or Same	(C or Z)
15-11		(reserved)	Always read as ‘0’

Actually only bit 3-0 is real bits that could be written to with LDi, LDip, LDp, bit 10-8 is just logic representation and the other bits are always read as ‘0’.

Map register:

Bit	**Function**	**Comment**
1-0	R74map	Selects register map for general purpose registers 7-4.
4	R168map	Selects register map for general purpose registers 16-8.

Due to possibilities when using LUT-based ram for register gp16-4 there is 4 sets of register gp7-4 and 2 sets of register gp16-8.
By changing the map bits, you get extra registers. Instead of 4 special registers (reg0-3) and just 12 general purpose registers; you actually have 4+32 registers!

Instructions

Instruction types:

Type	**Comment**	**Example**
Xkii	X=Instr, k=kreg, ii=immediate value	ADDi kreg,ii ADDi SP,3

Xkmg	X=Instr, k=kreg, m=mreg, g=displacement (0=none, E=postIncrement, F=preDecrement)	LDp kreg,*mreg
		LDp gp4,*gp15
		LDp gp4,*--gp5

XkmY	XY=instr, k=kreg, m=mreg	AND kreg,mreg AND gp5,gp7
XiiY	XY=instr, ii=immediate value	DH ii DH 0x45
XZZY	XYZ=instr	RET

Naming of instructions:
XXXi means that an immediate value is used in operation as a value
XXXp means that a pointer is used
XXXip means that an immediate value is used as a pointer in the operation

Instructions:

Hexcode	**Instruction**	**Comment**
00ii	CALLi ii	Call an address pointed to by ii value.
10ii	CALLRi ii	Call an relative address pointed to by PC+ii value
2---	(reserved)	
3kii	ADDi kreg,ii	Add ii to kreg
4kii	ANDi kreg,ii	And kreg with ii and store in kreg
5kii	ORi kreg,ii	
6kii	XORi kreg,ii	
7kii	LDi kreg,ii	Load kreg with value ii
A---	(reserved)	

B---	(reserved)	
Ckii	LDip kreg,ii	Load kreg with value at address ii
Dkii	STip ii,kreg	Store kreg at address ii

Ekmg	LDp kreg,*mreg
	LDp kreg,*--mreg
	LDp kreg,*mreg++
		Load kreg with the value pointed to by mreg. Either mreg could be post incremented, pre decremented or not changed at all by the operation.

Fkmg	STp *mreg,kreg STp *--mreg,kreg STp *mreg++,kreg	Store kreg at the address pointed to by mreg…
8km0- 8km2	(reserved)	
8km3	ADD kreg,mreg	Add mreg to kreg
8km4	AND kreg,mreg	
8km5	OR kreg,mreg	
8km6	XOR kreg,mreg	
8km7	MOV kreg,mreg	Move mreg to kreg
8ks8	ROR kreg	Rotate kreg
8ks9	RORC kreg	Rotate kreg thorugh carry flag
8ksA	LSR kreg	Logic shift kreg
8ksB	ASR kreg	Arithmetic shift kreg
8ksC	ROL kreg	
8ksD	ROLC kreg	
8ksE	LSL kreg	(Also ASL kreg)
8ksF	(reserved)	
9000	RET	Return from call
9ii3	DHi ii	Load high immediate value
9kp4	IFSET kreg,p-bit	If p-bit set in kreg, run next instruction else skip next instruction. If next instruction should be skipped and next instruction is a DHi instruction, then the instruction after DHi instruction is as well.
9kp5	ICLR kreg,p-bit	If p-bit cleared in kreg, run next instruction else skip next instruction. If next instruction should be skipped and next instruction is a DHi instruction, then the instruction after DHi instruction is as well.
9kp6	SETBIT kreg,p-bit	Set bit p in kreg
9kp7	CLRBIT kreg,p-bit	Clear bit p in kreg
9xxx (Other)	(reserved)	

Reserved instructions must not be used! They will do unpredictable things with registers, ram and other stuff… ☺

Explanation:

Letter(s)	**Type**	**Comment**
k	kreg	Store register for instructions
m	mreg	Read register for instructions
p	bit ptr	Bit pointer
s	bits to rotate	For now, always set to 1!
g	displacement	Displacement for pointers. 0=no displacement

		E=post increment pointer register after read/write to ram. F=pre decrement pointer register after read/write to ram.
ii	immediate value	Immediate value used for pointing and as a parameter for operator.

Alias:

Alias	**Encoded instruction**	**Comment**
JMPi ii	LDi PC,ii	PC=reg 0. Note: Might affect flags aswell.
JMPRi ii	ADDi PC,ii	See JMPii ii.
JMPR mreg	ADD PC,mreg	Register value relative jump
SET kreg	ORi kreg,-1	
CLR kreg	ANDi kreg,0	
NOT kreg	XORi kreg,-1	
PUSH kreg	STp *--SP,kreg	
POP kreg	LDp kreg,*SP++	
NOP	MOV 0,0	Move any register to itself
SUBi kreg,ii	ADDi kreg,-ii	
IFEQ	IFSET ST,Z	ST=reg2, Z=bit1
IFNE	IFCLR ST,Z	

Recommended address layout

After reset PC always point to address 0.
Address 0 should point to a PROM address with a jump instruction to program start.
Due to useful “side effects” mentioned in section “Immediate handling” I suggest that SRAM should be placed at top-most addresses.

For now, reserve address 4-7 for a jump instruction for future IRQ implementation.

Immediate handling

Immediate values could be used as values in operations.
If an instruction with immediate value is not preceded with a DHi instruction then the immediate value is sign extended.
If a DHi instruction precedes the immediate instruction, the immediate value from the DHi instruction is used as the high 8 bits of the resulting immediate value.
Example:

```
DHi 0x99
LDi gp4,0x80 // Loads general purpose register 4 with 0x9980
// From now on DH is “ignored”.
LDi gp5,0x90 // Loads general purpose register 5 with 0xFF90
LDi gp6,0x05 // Loads general purpose register 6 with 0x0005
LDi gp7,0x9A // Loads general purpose register 7 with 0xFF9A
LDi gp8,0x01 // Loads general purpose register 8 with 0x0001
DHi 0x30
LDi gp9,0xAA // Loads general purpose register 9 with 0x30AA
// From now on DH is “ignored”.
LDi gp10,0xBB // Loads general purpose register 10 with 0xFFBB
```

Some useful side effects of sign extension:
If jump tables are placed in address 4-127 then they could easily be used by CALLi 0 to 0x7F.
If storage tables in SRAM are placed in address 0xFF80 to 0xFFFF then they could easily be accessed by LDip/STip 0x80-0xFF without need of DHi usage.

Assembler

The assembler is rather simple, it doesn't handle macros.
The assembler handles the instructions and some special operands:

Construction	Function	Comment
:<label>	Set a label	
.ORG <address>	Set address	
.DW	Reserv an address for a word on an address	This doesn't actually do anything except reserves an area to store data in
.EQU <label>,<value>	Give a label a value	Could be used as address or constant
WH(<value or label>)	High part of value	Bit 15-8 of value. Usage: DHi WH(MyLabel)
WL(<value or label>)	Low part of value	Bit 7-0 of value. Usage: LDi PC,WL(MyLabel)
<value>	Decimal value	
0x<value>	Hexadecimal value	
0b<value>	Binary value	

Example of assembler code:

```
.EQU ioport,0x4008
.ORG 0
:ResetVector
        DHi WH(CodeStart)
        LDi PC,WL(CodeStart)
.ORG 0x40
:CodeStart
        LDi gp4,0x55
        DHi WH(ioport)
        STip WL(ioport),gp4 // Load 0x55 to address 0x4008
```

Future

There are a lot this CPU can't do, but on the other hand it's written with the goal to be compact in logic footprint.

In the future an IRQ handling logic would be nice I think.
Due to the possibility to change register maps, contents switching is done rather rapidly.