Appendix B - CPU16B instruction Set

B1. Introduction

The CPU16B is a 16-bit Von Neuman architecture processor with a single address space shared between data and instructions. The dual-port memory provided by FPGAs allows simultaneous instruction fetches and data manipulation giving the performance of a Harvard architecture machine. The instruction set uses 5 basic formats as shown in figure 1.

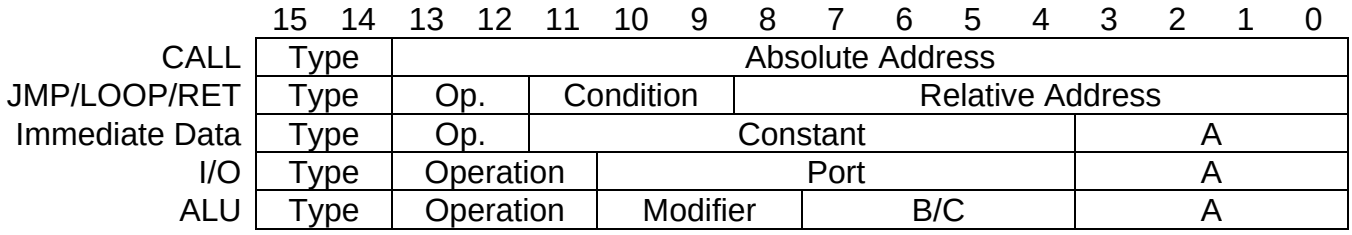

Figure 1 – Basic Instruction Format

Program-control instructions use two address formats. Call instructions use a 14-bit absolute address and can access 16,384 words of program memory. Jump instructions use a 9-bit relative address and can access the previous 255 or next 256 instruction locations. The jump is taken if the condition specified by a 3-bit field is satisfied. Otherwise instruction execution continues in sequence. The full program address space can be accessed by loop instructions that use a 14-bit absolute addresses stored by the mark instruction. This allows backward jumps to any location.

ALU instructions may operate on 16-bit words, the lower 4 or 8-bits of a word or any single bit. Most use register A as the source for one operand and the destination for the result. The second operand may be register B or a signed 8 or 16-bit constant.

I/O instructions use a 7-bit direct address to identify up to 128 16-bit wide I/O ports. Memory access instructions use 14-bit indirect addresses stored in register B plus an offset from the modifier field. Register A contains the data to be written to memory and is the repository for any data read from memory.

Assembler syntax uses 3 fields. The label field contains either blank space or an alphanumeric name followed by a colon. The operation field contains a 2-4-character instruction name. The operand field contains nothing, a single parameter or two to three parameters separated by commas.

```
Label: OP      Rd,Rs,mod    ; comments
```

Operands may be register names (SP and R1-R15), address labels or constants in binary, decimal, hexadecimal or alphanumeric format. A binary number starts with “#”, a hexadecimal number starts with “$” and a 1 or 2 character string is delimited by single quotes (‘).

B2. Program Flow Control

The program control instructions are listed in figure 2. Calls are unconditional and the call uses absolute addresses. The jump, loop and return instructions may be conditional or unconditional. Jump instructions use addresses that are relative to the program counter.

	15 14	13 12	11 10	9	8	7 6 5 4 3 2 1 0
CALL	00	Absolute Address				
JMP	01	00	Condition		Relative Address	
RET	01	01	Condition		0	
LOOP	01	01	Condition		1	
MARK	01	10	0	0		
STRA	01	10	0	1		
REP	01	10	1	*000*		Count

Figure 2 – Program Control Instructions

Three condition flags may be tested as shown in figure 3. The C, V and Z bits are the carry, overflow and zero flags for comparison operations and for 16-bit arithmetic operations, including add, subtract, increment and decrement. The C bit is also altered by bit test and shifting operations. The Z bit is necessary for loop control and the C bit is necessary for multi-word shifts and software multiplication and division routines.

Condition	Description
000	Always
001	Never (NOP)
010	V set by previous arithmetic or comparison instruction
011	V reset by previous arithmetic or comparison instruction
100	Z set by previous arithmetic or comparison instruction
101	Z reset by previous arithmetic or comparison instruction
110	C set by previous arithmetic, comparison, shift or bit test instruction
111	C reset by previous arithmetic, comparison, shift or bit test instruction

Figure 3 – Jump Conditions

The assembly language representation is an instruction name that specifies any condition code followed by an operand specifying the absolute address for calls or the relative address for jumps. Mark, return and loop instructions have no operand. Repeat has a single numeric operand.

JMP loads the program counter (PC) with the specified address and continues program execution from that location. JV, JNV, JZ, JNZ, JC and JNC jump if the V bit is true, the V bit is false, the Z bit is set, the Z bit is reset, the C bit is set or the C bit is reset. Otherwise, execution continues with the next instruction in sequence. The operand may be a label or a numeric value.

```
Here:  JMP   $0
       JNZ   there
```

CALL pushes the next instruction address onto the return address stack and then jumps to the specified address. The address is stored temporarily in a register while the stack pointer is incremented and then written into the dedicated RAM holding the return address stack. It contains up to 16 return addresses to allow subroutine nesting up to 16 levels. There is no overflow or underflow indication. . The operand may be a label or a numeric value.

```
Here:  CALL  $0
       CALL  There
```

The RET (return) instruction restores the address in the address stack to the program counter and continues execution from that location. RV, RNV, RZ, RNZ, RC and RNC return if the V bit is true, the V bit is false, the Z bit is set, the Z bit is reset, the C bit is set or the C bit is reset. The stack pointer is decremented after the address is read. There are no operands.

```
There: RET
```

The MARK instruction pushes the next instruction address onto the return address stack without jumping. It is used with the LOOP instructions for long backward jumps.

The STRA instruction pushes the contents of the selected register onto the return address stack. It is used with the RET instruction for indirect jumps.

```
       STRA R15
```

The LOOP instruction restores the address in the return address stack to the program counter and continues execution from that location. LV, LNV, LZ, LNZ, LC and LNC loop if the L bit is true, the V bit is false, the Z bit is set, the Z bit is reset, the C bit is set or the C bit is reset. The stack pointer is decremented if the loop is exited.

```
       LDL   R1,256
       MARK
       NOP           ; do this 256 times
       DEC   R1
       LNZ
```

The REP (repeat) instruction causes the next instruction to execute count + 2 times. It inhibits incrementing the program counter so the same instruction is issued multiple times with no additional overhead. The assembler calculates the correct value for the count field given the number of times to repeat the next instruction.

```
       REP   256
       NOP           ; do this 256 times
```

B3. Memory Access and I/O

Figure 4 shows the format of the immediate data, I/O and memory access instructions. They do not alter any flags.

	15 14	13 12	11 10 9 8	7 6 5 4	3 2 1 0
MVI	01	11	Data		A
LDH	10	101	101	Data	
WR	10	110	000	B	A
WRO	10	110	Offset	B	A
OUT	10	111	Port		A
IN	11	000	Port		A
RD	11	001	000	B	A
RDO	11	001	Offset	B	A

Figure 4 – I/O and Memory Access Instructions

The MVI instruction replaces the lower 8 bits of register A with immediate data and the sign is extended to the upper 8 bits. This can be modified by the LDH instruction that loads 8 bits of data into a special register. Those 8 bits replace the MSB in any MVI, ADI or CPI instruction that follows. The MVIW and LOAD pseudo-operations generate one LDH and one MVI instruction to load a 16-bit constant into a register. The LDA pseudo-operation does the same for loading a 14-bit address into a register.

```
MVI   #01111110
MVIW $3AB4
LDA   label
```

Output instructions use direct addressing to access up to 128 ports up to 16 bits wide. OUT copies the contents of register A to the selected port by placing the port address on IOADDR, the data on DOUT and asserting IOWR. Input instructions use direct addressing to access up to 128 ports up to 16 bits wide. IN copies the contents of the selected port from the DIN bus to register A and asserts IORD. IN takes 2 clock cycles to allow for propagation delays.

```
OUT   R1,48       ; numeric port number
IN    R1,data     ; port number defined by EQU
```

Memory access instructions use register indirect addressing of up to 65,536 words of memory. Writes complete in one clock cycle and reads take two cycles. Write with offset (WRO) copies the contents of register A to the memory location at the address in register B plus a 3-bit offset contained in the instruction. Read with offset (RDO) copies the contents of the memory location at the address in register B plus a 3-bit offset contained in the instruction to register A. The write (WR) and read (RD) instructions are the same but with a fixed offset of zero.

```
RDO   R2,R1,0     ; contents of memory at address in R1 copied to R2
RD    R2,R1       ; contents of memory at address in R1 copied to R2
WR    R3,R2       ; contents of R3 copied to memory at address in R2
```

B4. Arithmetic Operations

The 4, 8 and 16-bit arithmetic instructions are encoded as shown in figure 5. The carry, overflow and zero flags are updated for 16-bit arithmetic operations but not for 4-bit or 8-bit operations.

	15 14	13 12	11 10 9 8	7 6 5 4	3 2 1 0
ADI	10	00	C		A
INC	10	00	00000000		A
SBI	10	00	-C		A
DEC	10	00	11111111		A
CPI	10	01	C		A
CPZ	10	01	00000000		A
CMP	10	101	011	B	A
CLC	10	101	110	0000	*0000*
STC	10	101	110	0001	*0000*
CMPC	10	101	111	B	A
MOV	11	010	000	B	A
MVN	11	010	001	B	A
NEG	11	010	001	A	A
ADD	11	010	010	B	A
SUB	11	010	011	B	A
MOVC	11	010	100	B	A
MVNC	11	010	101	B	A
NEGC	11	010	101	A	A
ADC	11	010	110	B	A
SBC	11	010	111	B	A
MOV8	11	011	000	B	A
MVN8	11	011	001	B	A
ADD8	11	011	010	B	A
SUB8	11	011	011	B	A
MOV4	11	011	000	B	A
MVN4	11	011	001	B	A
ADD4	11	011	010	B	A
SUB4	11	011	011	B	A

Figure 5 –Arithmetic Instructions

ADI adds an 8-bit signed constant to register A (A=A+C). SBI generates the same instruction but negates the constant to subtract from register A (A=A-C). INC and DEC provide an alternate way of specifying ADI R_A, 1 and SBI R_A, 1. ADIW and SBIW prefix an LDH instruction for 16-bit operations.

```
ADI    R2,1   ; 2 ways to increment R2 by 1
INC    R2
```

CPI subtracts and 8-bit signed constant from register A and sets the zero and carry flags. No register is modified. CPZ provides a shorthand way of specifying CPI R_A, 0. CPIW generates an LDH followed by a CPI for 16-bit comparisons.

```
CPI   R1,0   ; 2 ways to compare R1 to zero
CPZ   R1
CPIW  R2,3456
```

STC and CLC set and clear the carry flag, respectively.

```
STC          ; set carry
CLC          ; clear carry
```

CMP subtracts register B from register A and sets the zero and carry flags. No register is modified. CMPC is the same, but propagates the carry bit for larger comparisons.

```
CMP   R2,R1        ; compare R2 to R1
```

MOV copies register B to register A. MVN copies the two's complement of register B to register A. MOVC and MVNC perform the same operations but propagate the carry bit. NEG and NEGC provide an alternate way to specify MVN R_N,R_N and MVNC R_N,R_N.

```
MOV   R2,R1        ; copy R1 to R2 and set flags
NEG   R2           ; negate R2
MVN   R2,R1        ; copy negative of R1 to R2
```

ADD adds register B to register A and leaves the result in register A (A=A+B). SUB subtracts register B from register A and leaves the result in register A (A=A-B). ADC and SBC are the same, but propagate the carry bit for larger arithmetic operations.

```
ADD   R1,R3        ; 32-bit add (R2,R1 + R4,R3)
ADC   R2,R4
```

MOV8 copies the contents of register B into the lower byte of register A. MVN8 negates the byte being copied. NEG8 provides an alternate way to specify MVN8 R_N,R_N.

```
MOV8  R2,R1        ; copy LSB of R1 to R2 without affecting flags
MVN8  R2,R1        ; same, but resultant byte is negated
```

ADD8 and SUB8 add or subtract the lower byte in B from the lower byte in A without affecting the upper byte ($A_{7-0} = A_{7-0} \pm B_{7-0}$ and $A_{15-8} = A_{15-8}$).

```
ADD8  R2,R1
SUB8  R2,R1
```

MOV4, MVN4, NEG4, ADD4 and SUB4 perform the same operations on the lower 4 bits without affecting the upper 12 bits.

B5. Logical Operations

The logic unit operates on two 16-bit inputs (A and B) or on one bit from input A. The encoding is shown in figure 6.

	15 14	13 12 11	10 9 8	7 6 5 4	3 2 1 0
NOT	11	100	000	B	A
AND	11	100	001	B	A
OR	11	100	010	B	A
XOR	11	100	011	B	A
MASK	11	100	100	C	A
RST	11	100	101	C	A
SET	11	100	110	C	A
INV	11	100	111	C	A

Figure 6 – Logical Instructions

Four instructions perform logical operations between register A and register B and four instructions perform logical operations on bits in register A.

NOT copies the one's complement of register B to register A. OR sets bits in register A when either of the corresponding bits in register A or register B are 1. XOR sets bits if only one of the two corresponding bits is 1. AND sets bits in register A if the corresponding bits in register A and register B are both 1.

```
NOT   R1,R1       ; R1 <- ~R1
OR    R2,R1       ; R2 <- R2 | R1
XOR   R3,R1       ; R3 <- R3 ^ R1
AND   R4,R1       ; R4 <- R4 & R1
```

MASK selects the lower 0-15 bits of register A by zeroing the upper bits.

```
MASK  R6,8  ; zero upper byte of R0
```

The bit manipulation instructions modify register A. RST and SET change the value of the bit selected by C to 0 and 1, respectively. INV complements the value of the selected bit.

```
RST   R7,1   ; clear bit 0
SET   R7,15 ; set bit 15
INV   R7,7   ; invert bit 7
```

B6. Shift, Rotate and Sign Extension Operations

Shift and rotate operations are implemented by changing the order of bits in registers. In addition, individual bits can be copied into the carry bit. Jumps can then be made conditional on the value of that bit. The sign bit on 8-bit bytes may also be extended to 16-bit words.

Figure 7 shows how these instructions are encoded. Note that additional operations are available by using different values for bits 7-4 in the instruction. The most useful instructions are documented here.

	15 14	13 12 11	10 9 8	7 6 5 4	3 2 1 0
TST	10	101	100	C	A
MLLS	11	101	000	B	A
SXL	11	101	000	A	A
MHLS	11	101	001	B	A
SXH	11	101	001	A	A
INS4	11	101	010	B	A
ROR4	11	101	010	A	A
INS8	11	101	011	B	A
SWAP	11	101	011	A	A
IN16	11	101	100	B	A
ROL4	11	101	100	A	A
REV	11	101	101	B	A
ROR	11	101	110	0000	A
ROL	11	101	111	0101	A
RRC	11	101	110	1000	A
RLC	11	101	111	1001	A
LSR	11	101	110	1100	A
LSL	11	101	111	1101	A
ASR	11	101	110	0100	A
SHL	11	101	110	C	A
SHR	11	101	111	C	A

Figure 7 – Shift and Rotate Instructions

TST copies bit C of register A to the carry flag.

```
TST   R5,4   ; copy bit 4 to carry flag
```

MLLS copies the least significant byte of register B to the least significant byte of register A and then extends the sign in A to fill 16 bits. MHLS copies the most significant byte of register B to the least significant byte of register A and then extends the sign in A to fill 16 bits. SXL and SXH provide alternate ways of specifying MLLS R_A, R_A and MHLS R_A, R_A.

```
MLLS  R3,R1 ; split word in R3 into LSB in R1 and MSB in R2
MHLS  R3,R2
SXL   R3    ; remove upper byte in R0 and sign extend lower byte
```

INS4 copies the lower 4 bits of register B to the upper 4 bits of register A after shifting the upper 12 bits of register A to the right. INS8 copies the lower 8 bits of register B to the upper 8 bits of register A after shifting the upper 8 bits of register A to the right. INS4 copies the lower 12 bits of register B to the upper 12 bits of register A after shifting the upper 4 bits of register A to the right. ROR4, SWAP and ROL4 provide alternate ways of specifying INS4 R_A, R_A, INS8 R_A, R_A and IN12 R_A, R_A.

```
INS4   R1,R2 ; copy LS 4 bits of R2 to bits 15-12 of R1 and shift bits 15-4 to 11-0
ROR4 R1      ; rotate R0 right by 4 bits
```

REV reverses the order of the bits while copying from register B to register A. Bits 15 and 0 are swapped, bits 14 and 1 are swapped, etc.

```
LDL    R1,$57
REV    R1,R1        ; 01010111 -> 11101010
```

ROL and ROR rotate the contents of register A left or right by one bit with bit 15 replacing bit 0 or bit 0 replacing bit 15, respectively. RLC and RRC rotate the contents of register A left or right by one bit with the carry bit replacing bit 0 or bit 15, respectively. The carry bit contains the previous value of bit 15 after ROL or RLC and bit 0 after ROR or RRC.

```
LOAD R1,$F0       ; R1=11110000
ROR   R1          ; R1=01111000, C=0
ROL   R1          ; R1=11110000, C=0
RLC   R1          ; R1=11100000, C=1
RRC   R1          ; R1=11110000, C=0
```

LSL shifts the contents of register A left by one bit and clears bit 0 while setting the carry bit to the previous value of bit 15. LSR shifts the contents of register A right by one bit and clears bit 15 while setting the carry bit to the previous value of bit 0.

```
LOAD R2,$FF       ; R2=11111111
LSL   R2          ; R2=11111110, C=1
LSR   R2          ; R2=01111111, C=0
```

ASR shifts the contents of register A right by one bit without affecting bit 15. The carry register contains the previous value of bit 0.

```
LOAD R1,-2        ; R1=11111110
ASR   R1          ; R1=11111111 (-1), C=0
```

The SHL and SHR instructions provide a means to generate additional types of 1-bit shifts as shown below:

C	Carry Flag	LSB or MSB
00x0	Bit 0	Bit 0
00x1	Bit 15	Bit 0
01x0	Bit 0	Bit 15
01x1	Bit 15	Bit 15
10x0	Bit 0	Carry flag
10x1	Bit 15	Carry flag
1100	Bit 0	0
1101	Bit 15	0
1110	Bit 0	1
1111	Bit 15	1

Figure 8 – SHL and SHR Modifier Field Encoding

The value of the bit shifted into the LSB during a left shift (SHL) or into the MSB during a right shift (SHR) may be obtained from the carry flag or fixed at 0 or 1. Either the most or least significant bit can be copied to the carry flag.

B7. Optional Multiply-Accumulate Instructions

There are 10 multiply-accumulate instructions, as listed in figure 9. Type 2 instructions are used to start multiplication with options selected by the modifier field. The instructions complete in 1 cycle but results require 2 instruction cycles to become available. Type 3 instructions are used to copy the results into general-purpose registers.

	15 14	13 12 11	10 9 8	7 6 5 4	3 2 1 0
UMUL	10	100	000	B	A
UMLN	10	100	001	B	A
UMAC	10	100	010	B	A
UMSB	10	100	011	B	A
MUL	10	100	100	B	A
MULN	10	100	101	B	A
MAC	10	100	110	B	A
MSUB	10	100	111	B	A
LPL	11	110	000	*0000*	A
LPH	11	110	001	*0000*	A

Figure 9 – Multiply Instructions

MUL performs a 16-bit by 16-bit signed multiply and leaves the result in a 32-bit accumulator. UMUL performs an unsigned multiply. MULN and UMLN perform signed and unsigned multiplies and negate the result.

MAC and UMAC perform signed and unsigned multiplies and add the result to the existing accumulator contents. MSUB and UMSB perform signed and unsigned multiplication and subtract the result from the accumulator. Operands may be introduced at a rate of one pair per instruction cycle and the results will be available on the next instruction cycle after the final operands are loaded.

LPH provides the upper 16-bits of the accumulator and LPL provides the lower 16 bits of the accumulator.

The MAC unit is useful for complex multiplies. $C_I = A_IB_I - A_JB_J$ can be implemented with a MUL instruction followed by MSUB, LPH and LPL instructions. $C_J = A_IB_J + A_JB_I$ can be implemented with a MUL instruction followed by MAC, LPH and LPL instructions. The following code fragment performs a complex multiply on R1/R2 and R3/R4 and returns the result in R5-R8.

```
MUL     R1,R3     ; AIBI
MSUB    R2,R4     ; AIBI - AJBJ
LPL     R5
LPH     R6
MUL     R1,R4     ; AIBJ
MAC     R2,R3     ; AIBJ + AJBI
LPL     R7
LPH     R8
```

B8. Optional Division Instructions

There are 4 division-related instructions that are formatted as shown in figure 10. The division operations consume many clock cycles, but may occur in parallel with other operations. The programmer must insure that a new division operation is not issued until the previous one has been completed.

	15 14	13 12 11	10 9 8	7 6 5 4	3 2 1 0
FDIV	10	101	000	*0000*	*0000*
IDIV	10	101	001	B	A
LQ	11	110	010	*0000*	A
LR	11	110	011	*0000*	A

Figure 10 – Division Instructions

IDIV performs a 16-bit by 16-bit unsigned integer division leaving a 16-bit quotient and 16-bit remainder. A is divided by B and the result is left in the quotient and remainder registers after 17 clock cycles.

FDIV continues the calculation in order to generate the fractional portion of the quotient in Q after 17 more clock cycles.

The LQ and LR instructions read the quotient and remainder registers. The upper bit of R may be monitored to determine when the calculation is complete. It is 1 during computation.

```
LOAD R2,255
LOAD R1,4
IDIV  R2,R1       ; 255 ÷ 4
REP   17
NOP
LQ    R1          ; 63
LR    R2          ; 3
FDIV              ; 3 ÷ 4
REP   16
NOP
LQ    R3          ; R3 = 1100000000000000 = ¾
```

B9. Stack Operations

The register R0 may be specified by SP and is commonly used as a stack pointer for data storage. The assembler will generate 2-instruction sequences for PUSH and POP as follows:

```
DEC   SP          ; PUSH R_N
WR    R_N,SP

RD    R_N,SP      ; POP R_N
INC   SP
```

By default, the use of R0 is suppressed to minimize register allocation errors.

B10. Pseudo-Instructions

The PRAM instruction is used to accommodate different types and amounts of program memory. UCF files are used to initialize program memory when the FPGA is configured. Hexadecimal format files may be downloaded after FPGA configuration.

Modifier	Result
0	Generate hex file (any length)
1	UCF file for one 1k x 16 RAM
2	UCF file for two 2k x 8 RAMs
4	UCF file for four 4k x 4 RAMs

Figure 11 – PRAM Instruction

REG assigns a label to the specified register and EQU assigns a label to the specified numeric constant.

```
Label  REG   R7
one    EQU   1
```

DW and DB initialize the value of a word or the 2 bytes within a single word, respectively.

```
       DW    28
       DB    37,48
```

DS assigns a label to the current address and increments the data address by the specified number of words.

```
long:  DW    2
```

The ORG pseudo-instruction is used to set the program address to the specified value. The program address defaults to zero.

```
       ORG   0
```

B11. Summary

Figure 12 shows how instructions are allocated within the space provided by the type, operation and modifier fields. All instructions are 16 bits and all except IN, RD, IDIV and FDIV execute in one clock cycle.

CPU16B Operation Code Matrix

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	CALL															
1																
2																
3																
4	JMP		NOP		JV		JNV		JZ		JNZ		JC		JNC	
5	RET	LOOP	NOP	NOP	RV	LV	RNV	LNV	RZ	LZ	RNZ	LNZ	RC	LC	RNC	LNC
6	MARK				STRA				RPT							
7	MVI															
8	ADI															
9	CPI															
A	UMUL	UMLN	UMAC	UMSB	MUL	MULN	MAC	MSUB	FDIV	IDIV		CMP	TST	LDH	CL/STC	CMPC
B	WR								OUT							
C	IN								RD							
D	MOV	NEG	ADD	SUB	MOVC	NEGC	ADC	SBC	MOV8	NEG8	ADD8	SUB8	MOV4	NEG4	ADD4	SUB4
E	NOT	AND	OR	XOR	MASK	RST	SET	INV	SXL	SXH	ROR4	SWAP	ROL4	REV	SHR*	SHL*
F	LPL	LPH	LQ	LR												

*Shift instructions include variants ROR, ROL, RRC, RLC, LSR, LSL and ASR.

Figure 12 – Operation Code Matrix

JBS