1 The ARC processor (data path and control)

1.1 Introduction

This document is a summary of the data path and micro controlled controller for the ARC processor (Chapter 5 of Computer Architecture and Organization; An Integrated Approach, Murdocca and Heuring, ISBN 978-0-471-73388-1)

A VHDL model is available: http://iiusatech.com/murdocca/CAO/VHDL.html

1.2 Main part of data path

Figure 5-3 (MH[1]) shows the main part of the data path. Two registers from the register file can be read at the same time (A bus and B bus), and one register can be written (C bus).

The user has, via an instruction, access to the registers %r0 until %r31. For handling an instruction and the order of execution of instructions additional registers are available:

%r32 (PC)	program counter; points to the (next) instruction in main memory
%r33-%r36:	temporary registers; are available for internal execution of an instruction
%27 (IR)	instruction register; contains the instruction that is executed (the controller can read this)

To address one of the 38 registers 3 decoders with 6 input lines are required (A, C and C decoder).

The ALU operations are in Figure 5-4 (MH). The SCC (Set Condition Codes) is true when F3=0 and F2=0. In that case the PSR (Processor Status Register) is updated with the output n, z, v an c. Hence, SCC is connected to the enable input for the flip flops that store the flags!

➔if a micro-instructions changes the flags, then the new status information is available in the next clock cycle!

An ALU instruction uses only input A or both inputs. E.g. INCPC(A) adds 4 to the data that is on input A (the data on B bus is not used).

1.3 Data path

The left part of figure 5-10 (MH) shows the data path. The scratchpad contains the register file with the three decoders (figure 5-3 (MH)). Only the instruction register (%37, IR) is explicitly shown since its content is used in several parts of figure 5-10(MH).

A user can select a register, with an instruction, but also the controller can select a register from the register file. Therefore the input of the A Decoder (in the scratchpad) is connected with a multiplexer (A Mux). The select input is controlled by the controller. Note: the 5 bits register address (part of the instruction) is extended with a 0 (only the lower 32 addresses can be accessed by the user).

Also for the B decoder and de C decoder additional multiplexers are address, resp. B Mux and C Mux.

A fourth multiplexer is added that is connected with the data input of the scratchpad. With this multiplexer the output of the ALU is selected or the data out from main memory. This multiplexer is also controlled by the controller.

[1] MH is used if it is a figure in the book of Murdocca and Heuring.

1.4 Controller

The micro programmed controller is in the right part of figure 5-10(MH). De control store contains the micro instructions.

1.4.1 Micro instruction format

The micro instruction that is executed is in the MIR register. Figure 5-11(MH) shows the micro instruction format:

Field A: With these 6 bits one of the 38 registers in the Scratchpad is selected. The contents of the selected register is on the A bus (input of ALU). However only when field AMUX is 0.

Field AMUX: This bit is connected to the multiplexer A MUX (Figure 5-10(MH)). When this bit is 0 field A is input of the A Decoder (figure 5-3(MH)). When this bit is 1 field rs1 of the instruction is zero extended and input of the A decoder.
Hence: with this bit the controller can decide to use the address in the instruction or the address that is in MIR.

Fields B, BMUX, C, CMUX: have similar behavior as the fields A and AMUX.

Field RD: When this field is 1 data is read form main memory.
The A bus is connected with the address bus of the main memory. The data out pins of the main memory is connected with multiplexer "C Bus MUX", and since RD is 1 this is data is on the C bus (figure 5-3(MH)).

Field WR: When this field is 1 data is written in main memory. Address bus of the main memory is connected with the A bus, and the data In bus if the main memory is connected with the B bus.

ALU: With these 4 bits one of the ALU operations is selected from figure 5-4(MH).

COND: A micro instruction is stored on one address in the control store. In most cases the next micro instruction that is executed is on the next control store address. However it is possible to jump to another address (e.g. change the order of execution of the micro instructions). The is controlled with this field (see figure 5-12(MH)).

JUMP ADDR: If the condition (COND) is true a jump is made to this control store address.

1.4.2 Fetch, decode and execute cycle

The operation of the controller is illustrated with an example.

Assume that main memory at address 200_{10} the following 32 bits are stored

1	0	0	0	1	1	1	0	1	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Is this an instruction? It is not known! It could also be data.

Now assume that the program counter (%r32 in the register file, figure 5-3(MH)) contains 200_{10}. The next instruction to be executed is on this address, therefore:

1. Fetch: The data at this main memory address is stored in register in the instruction register (%r37 in the register file, figure 5-3(MH)).
2. Decode: The controller determines the instruction.
3. Execute: The instruction is executed
4. Increment program counter with 4 (since each instruction is 32 bits, and in main memory each address contains 8 bits), or jump to a different address (branch instruction and the branch condition is true).

The first two steps (fetch and decode) are the same for all instructions.

1.4.2.1 Micro instruction fetch

The program counter (PC) contains the address in main memory with the instruction to be executed. Hence:

1. The contents of PC should be on the A bus (this bus is connected with the address bus of main memory).
2. RD signal (in the micro instruction) should be 1.
3. The data read from main memory should be stored in the register file at address %r37 (instruction register).

The micro instruction that fetches the instruction is in control store address 0. This RTL description that is used by the authors is a strange (at first sight).

R[IR]← AND(R[PC], R[PC]) ; READ Figure 5-15(MH)

The AND operation refers to the AND operation of the ALU. It seems that the logical AND is taken from R[PC] and R[PC] and that the result is stored in R[IR]. However, due to READ (RD is 1) the "C Bus MUX" (figure 5-10(MH)) will select the data out of main memory instead of the output of the ALU. But why the AND operation? Well, any ALU operation, since the result of the operation is not used, can be used that uses the A bus.

The micro instruction for fetch is (symbolic and the actual bit pattern)

Symbolic

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
0	PC	0	PC	0	IR	0	1	0	ADD	NEXT	-------

Bit pattern

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
0	100000	0	100000	0	IR	100101	1	0	1000	000	00000000000

Note: for the don't care of the Jump Addr field zeros are used in the book. In the remainder we will use don't cares.

Since field COND is 000 (NEXT) the controller will select the micro instruction on the next address in the control store (details are discussed in a minute).

1.4.2.2 Micro instruction decode

		rd					op3						rs1					i	simm13												
1	0	0	0	1	1	1	0	1	0	0	0	0	0	0	1	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

In the decode step the instruction is determined. The instruction format is in figure 5-2(MH). Based on the left two bits (yellow) it is an “arithmetic instruction”, field op3 is “010000” hence it is instruction ADDCC.

How is the instruction ADDCC implemented in the controller? First have a look at the organization om the control store is:

Address (decimal)	Content
0	Fetch
1	Decode
2 1023	
1024 Area with the start addresses of instructions 2046	
2047	Increment PC with 4

The number of address bits is 11. The start address for the execution of an instruction in the upper half of the control store. Hence, this address begins with a 1: 1xxxxxxxxxx.
In most cases multiple micro instructions are required for the execution of an instruction. The designers of the ARC decided that 4 consecutive control store addresses should be reserved for an instruction. So if the start address of an instruction is: 1xxxxxxxx00 than 1xxxxxxxx01, 1xxxxxxxx10 and 1xxxxxxxx11 are available for that instruction. Now each instruction is uniquely determined by <op> field (left 2 bits) and <op3> (6 bits). These 8 bits are placed on field fields marked with xxxxxxxx. See figure 5-13(MH).
Note: some instruction are unique with <op> and <op2>. In that case bit 21..19 or 000.

Now the start address of the instruction ADDC is known:
1 10 010000 00 = 1600_{10}

How is this realized in the controller? In figure 5-10(MH) you see on top of the control store an “CS Address MUX”. This multiplexer has 3 data inputs (each 11 bits). With two bits (on the right of the multiplexer) one of the inputs are selected. The 3 data inputs are:

1. incremented value of the current control address
2. start address of an instruction: 1<8 bits>00 (as described above)
3. the JUMP ADDR from the current micro instruction.

The two selection bits for the multiplexer are output of the Control Branch Logic. The behavior of the this logic is in figure 5-12(MH). E.g.

1. if COND is 000 (=NEXT) than the output is 00
2. if COND is 111 (=DECODE) than the output is 10
3. if COND is 001 (=Negative) than if the status flag N (in PSR) is true the output is 01 else 00

The micro instruction for DECODE is (symbolic and the actual bit pattern)

Symbolic

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
1	-	-	-	-	%r0	0	0	0	-	DECODE	-

Bit pattern

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
1	-	-	-	-	000000	0	0	0	-	111	-

Note: Fields C and Cmux are not don't care! Since no data is to be changed in the register file data is written in register %r0. Since this register is always zero nothing is changed. Furthermore Rd and Wr are signal for the external main memory, therefore these signals are 0.

1.4.2.3 Execution of ADDCC

Next the micro program for instruction ADDCC has to be written. Often different programs are possible to realize the behavior of an instruction.

In figure 5-15(MH) on micro address 1600 until 1603 a RTL description is given:

1600: IF R[IR[13]] THEN GOTO 1602;	An arithmetic instruction has 2 formats: a) addcc %rs1, %rs2, %rd b) addcc %r1, sext13(R[ir]), %rd If IR[13] is 1 b) is chosen
1601: R[rd]←ADDCC(R[rs1], R[rs2]); GOTO 2047;	Format a), and goto 2047
1602: R[%temp0]←SEXT13(R[ir]);	The lower 13 bits are sign extended and stored in R[%temp0]
1603: R[rd]←ADDCC(R[rs1], R[%temp0]); GOTO 2047;	

The bit pattern (content of control store) for this program is in figure 5-17 (page 172).

An alternative, less trivial but less clock cycles (faster) is

1600: R[%temp0]←SEXT13(R[ir]); IF R[IR[13]] THEN GOTO 1602;	In this clock cycle %temp0 is already used to store SEXT13 lower 13 bits.
1601: R[rd]←ADDCC(R[rs1], R[rs2]); GOTO 2047;	
1602: R[rd]←ADDCC(R[rs1], R[%temp0]); GOTO 2047;	

Symbolic

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
1600	-	-	-	-	%temp0	0	0	0	SEXT13	Bit13	1602
1601	-	1	-	1	-	1	0	0	ADDCC	JMP	2047
1602	-	1	%temp0	0	-	1	0	0	ADDCC	JMP	2047

Bit pattern

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
1600	-	-	-	-	100001	0	0	0	1100	101	11001000010
1601	-	1	-	1	-	1	0	0	0011	110	11111111111
1602	-	1	100001	0	-	1	0	0	0011	110	11111111111

Notes:

1) Check that when Cmux is 0 the address in field C is used, and when Cmux is 1 field C is not used (the address in the instruction is used).
2) The start address of ADDCC is 1100100000**00**, the next possible start address is 1100100001**00**. What if a micro program needs more than 4 micro addresses? In that case you can jump to the lower half of the control store.

3) A fragment of a micro program is given below. %r0 and %temp1 are added and the status is set. Maybe the intention of the programmer (of this micro program!) was to continue on address 1810 when the addition results in a negative number (JMP neg). However you should remember that status is stored in a register (PSR, figure 5-10(MH)). Hence, when the micro instruction on address 1801 is executed the PSR does not yet updated with the new status. To implement the intention of the programmer an alternative is given. It depends on the problem if something useful can be done at address 1802 beside the JMP.

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
1801	%r0	1	%temp1	1	-	0	0	0	ADDCC	JMP neg	1810
1802											

Alternative

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
1801	%r0	1	%temp1	1	-	0	0	0	ADDCC	NEXT	-
1802										JMP neg	1810

1.4.2.4 Increment program counter

After an instruction is executed the program counter has to be updated. In case the instruction is a branch instruction than the program counter is already updated when it should make a jump. If

After the execution of an instruction is completed the program counter is incremented. For branch instructions the program counter is already updated as part of the execution. Most instructions jump to micro address 2047 where the program counter is incremented with 4 (each instruction is 32 bits, and a byte organized memory)

The micro instruction for incrementing the program counter with 4.

Symbolic

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
2047	%pc	0	-	-	%pc	0	0	0	INPC	JMP	0

Bit pattern

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
2047	100000	0	-	-	100000	0	0	0	1110	110	00000000000

Note: after incrementing the next micro address is 0 where the new instruction is fetched.

1.5 Exercises

Exercise 1:

The ARCTool supports the instruction XOR. However no micro program for this instruction is given in chapter 5.

a) What is the start address in the control store for this instruction.
   Hint: assembly a program with this instruction.
b) Give a micro program for XOR (symbolic and bit patterns; use – for fields that are don't care).

a)

```
00000004  0000000004  82180001                xor %r0, %r1, %r1
```

Machine code is
Hex:
82180001
Binary:
1000 0010 0001 1000 0000 0000 0000 0001
The relevant fields for the start address are marked in red (figure 5-2(MH). Figure 5-13(MH)).
The start address is
$1100000011_{00} = 1548_{10}$

b)

Now you have to realize A xor B with the available ALU operations.

Algorithm 1

$Y \leftarrow A \oplus B$
Is equal to
$Y \leftarrow A.\bar{B} + \bar{A}.B$
This can be realized with the following sequence of microinstructions:

```
R[%temp0]←R[%R0] orn R[B]          ; %temp0 is NOT R[B]
R[%temp1]←R[%R0] orn R[A]          ; %temp1 is NOT R[A]
R[%temp2]←R[A] AND R[%temp1];      ; %temp2 is A.NOT(B)
R[%temp3]←R[%temp0] AND R[B];      ; %temp3 is NOT(A).B
R[rd]← R[%temp2] AND R[%temp3];
```

5 clock cyles are required.

Algorithm 2

$Y \leftarrow A \oplus B$
Is equal to:
$Y \leftarrow (A + B).\overline{A.B}$
This can be realized with the following sequence of microinstructions:

```
R[%temp0]←R[A] or R[B]
```

R[%temp1]←R[A] and R[B]
R[%temp2]←R[%r0] orn R[%temp1] ; not(A.B)
R[rd]←R[%temp0] and R[%temp2]

4 clock cycles are required (faster!)

Algorithm 2 has less clock cycles (faster).
The whole micro program, including the two modes for operand B, is

RTL

1548: R[%temp3]←SIMM13(R[ir]); IF R[IR[13]] THEN GOTO 1550;	In this clock cycle %temp0 is already used to store lower 13 bits (zero extended).
1549: R[%temp3]←or(R[%r0], R[rs2]);	
1550: R[%temp0]←or(R[rs1], R[%temp3]);	
1551: R[%temp1]←and(R[rs1], R[%temp3]); GOTO 400	After 4 addresses jump to lower halve of memory (1552 can be a start address of an instruction). It is assumed that address 400 is available.
400: R[%temp2]←orn(R[%r0], R[%temp1]);	
401: R[rd]←and(R[%temp0], R[%temp2]);	

Symbolic micro program

Address	A	Amux	B	Bmux	C	Cmux	Rd	Wr	ALU	Cond	Jump addr
1548	-	-	-	-	%temp3	0	0	0	Simm13	Bit13	1550
1549	%r0	0	-	1	%temp3	0	0	0	or	nxt	-
1550	-	1	%temp3	1	%temp0	0	0	0	or	nxt	-
1551	-	1	%temp3	1	%temp1	0	0	0	and	jmp	400
400	%r0	0	%temp1	0	%temp2	0	0	0	orn	nxt	-
401	%temp0	0	%temp2	0	-	1	0	0	and	jmp	2047