Computer Architecture and Organization
Lecture 6

Datapath and controller of subset ARC

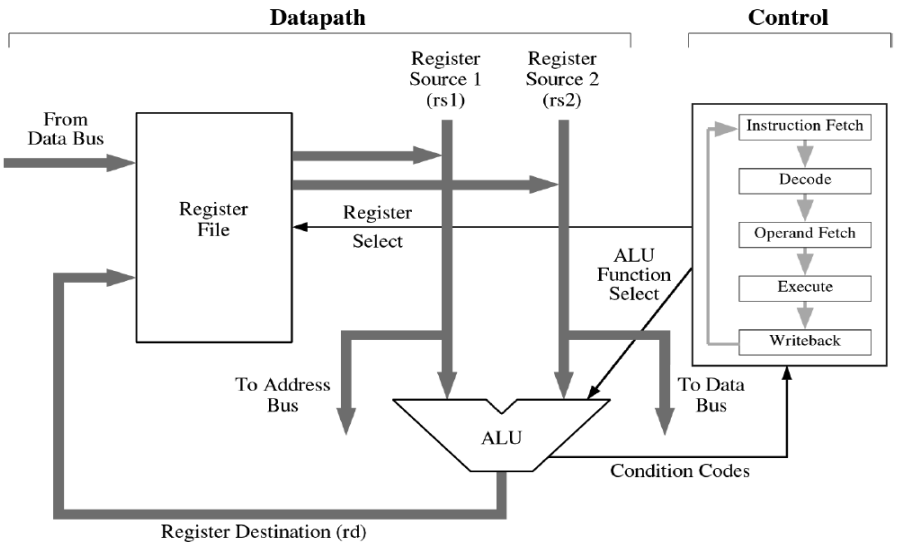
Learning objectives:

- You understand the details of the data path and controller
- You can write a micro program

Topics of this lecture

- Datapath ARC ⇦
- Controller ARC
  - Micro controlled

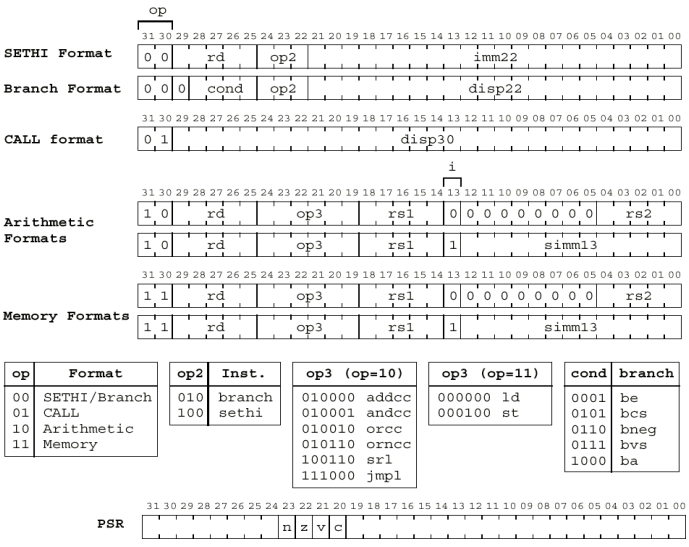
A More Detailed View



ARC Instruction Subset

	Mnemonic	Meaning
Memory	ld	Load a register from memory
	st	Store a register into memory
Logic	sethi	Load the 22 most significant bits of a register
	andcc	Bitwise logical AND
	orcc	Bitwise logical OR
	orncc	Bitwise logical OR of rs1 and the inverse of rs2
	srl	Shift right (logical)
Arithmetic	addcc	Add
Control	call	Call subroutine
	jmpl	Jump and link (return from subroutine call)
	be	Branch if equal
	bneg	Branch if negative
	bcs	Branch on carry
	bvs	Branch on overflow
	ba	Branch always

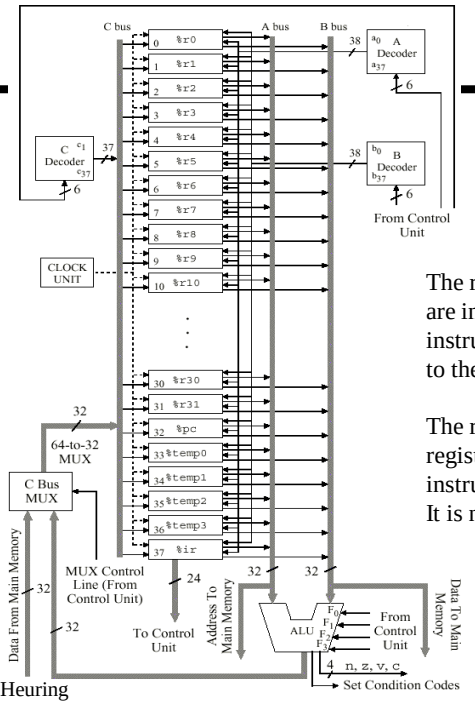
ARC Instruction Formats



Topics of this lecture

- Datapath ARC
- Controller ARC⇦
  - Micro controlled

ARC Datapath

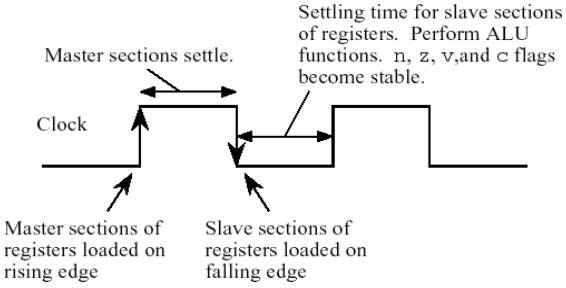


The registers %temp0-%temp3 are internally used by an instruction and are not visible to the user.

The register %ir (instruction register) holds the current instruction that is being executed. It is not visible to the user.

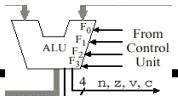
Timing Relationships for the Registers

- **Notice that the book uses the following timing relation**



- **We will adopt the style of a synchronous system with memory element active on the falling edge of the clock (*D-flipflop*).**

ALU



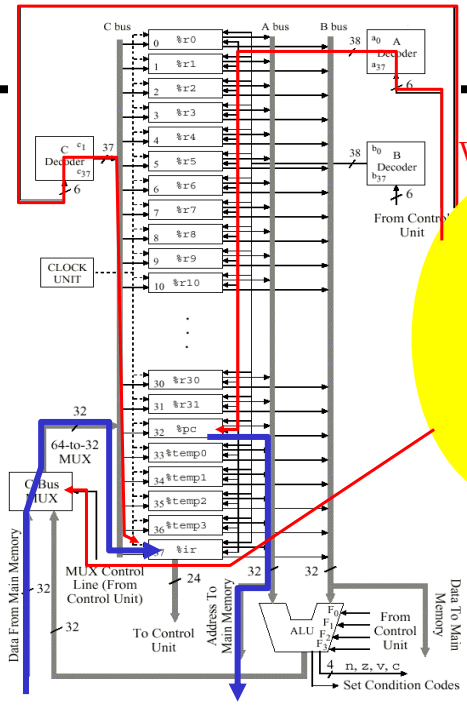
F_3	F_2	F_1	F_0	Operation	Changes Condition Codes	
0	0	0	0	ANDCC (A, B)	yes	
0	0	0	1	ORCC (A, B)	yes	
0	0	1	0	ORNCC(A, B)	yes	A OR NOT B
0	0	1	1	ADDCC (A, B)	yes	
0	1	0	0	SRL (A, B)	no	Shift Right Logical A over B positions
0	1	0	1	AND (A, B)	no	
0	1	1	0	OR (A, B)	no	
0	1	1	1	ORN(A, B)	no	
1	0	0	0	ADD (A, B)	no	
1	0	0	1	LSHIFT2 (A)	no	ShiftLeft A over 2 positions
1	0	1	0	LSHIFT10 (A)	no	ShiftLeft A over 10 positions
1	0	1	1	SIMM13 (A)	no	Unsigned extention lower 13 bits
1	1	0	0	SEXT13 (A)	no	Signed extention lower 13 bits
1	1	0	1	INC (A)	no	A+1
1	1	1	0	INCPC (A)	no	A+4
1	1	1	1	RSHIFT5 (A)	no	Shift Right A over 5 positions with sign extension

Example: instruction addcc %r4,%r5,%r6

- **Bit pattern of instruction addcc %r4,%r5,%r6 is**
  10 00110 010000 00100 0 00000000 00101

arithmetic rd=%r6 op3=addcc rs1=%r4 bit13=0 ➔register rs2=%r5

Instruction fetch



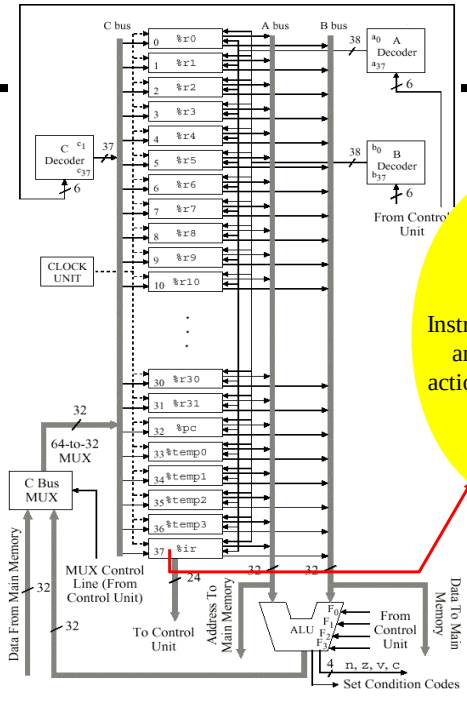
Write in %ir;

Controller

RD (read memory)

M[%pc]

Decode

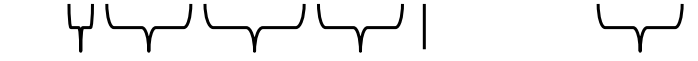


Controller

Instruction is known and appropriate actions can be taken

Decoding of addcc %r4,%r5,%r6

- **Bit pattern of instruction addcc %r4,%r5,%r6 is 10 00110 010000 00100 0 00000000 00101**



arithmetic rd=%r6 op3=addcc rs1=%r4 bit13=0 ➔register rs2=%r5

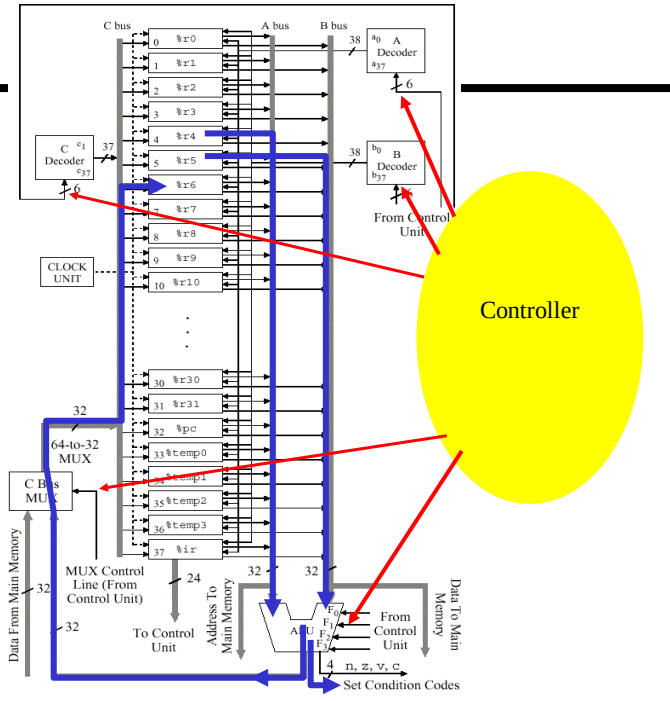
bit13 determines if immediate or register value is used.
The controller instructs the datapath:

1. On A bus contents of %r4 (A decoder)
2. On B bus contents of %r5 (B decoder)
3. Instruct ALU to perform add operation
4. C mux should select output ALU
5. Result should be stored in %r6 (C decoder)

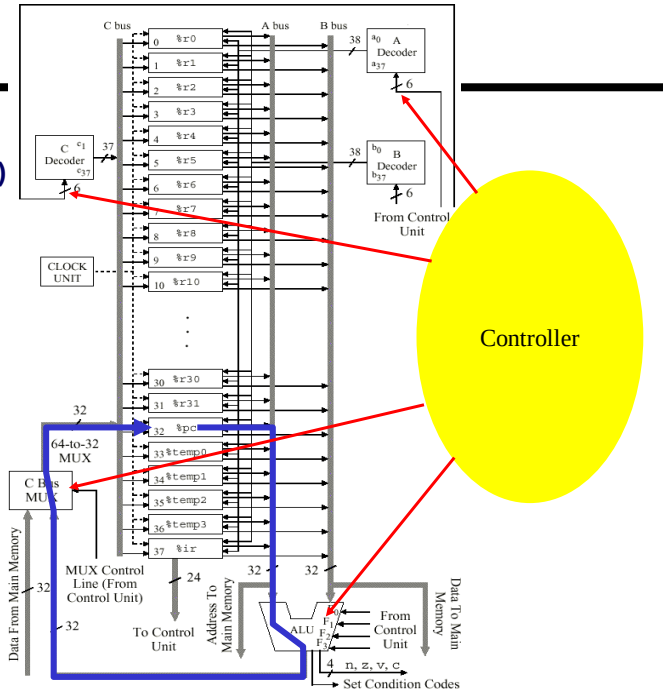
Operand fetch

Execute

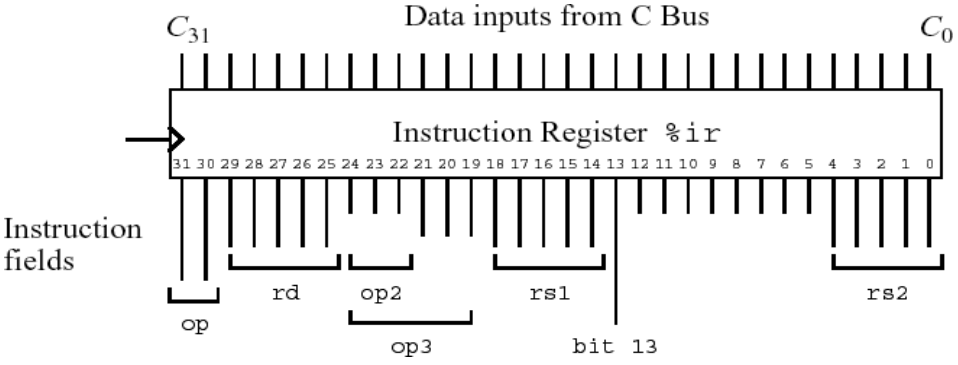
Write back



Increment PC
(Program Counter)



Outputs to Control Unit from register %ir



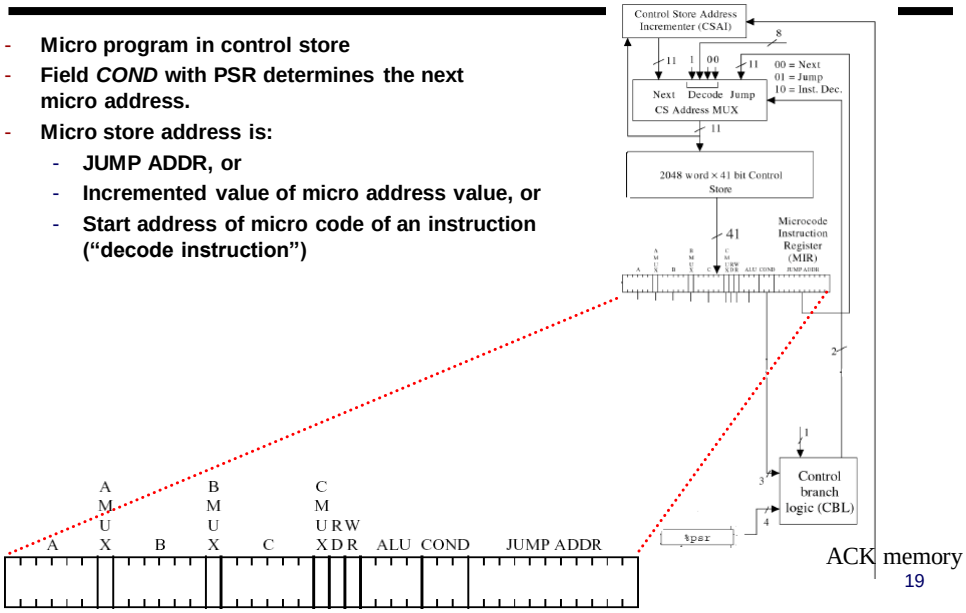
op, op2/op3 and bit13 are used by the controller

Topics of this lecture

- Exercises
- Datapath ARC
- Controller ARC ⇦
  - Micro controlled

Micro programmed controller

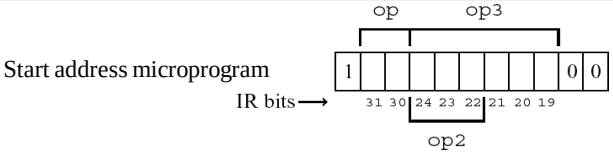
- Micro program in control store
- Field *COND* with PSR determines the next micro address.
- Micro store address is:
  - JUMP ADDR, or
  - Incremented value of micro address value, or
  - Start address of micro code of an instruction ("decode instruction")



Settings for the COND Field of the Microword

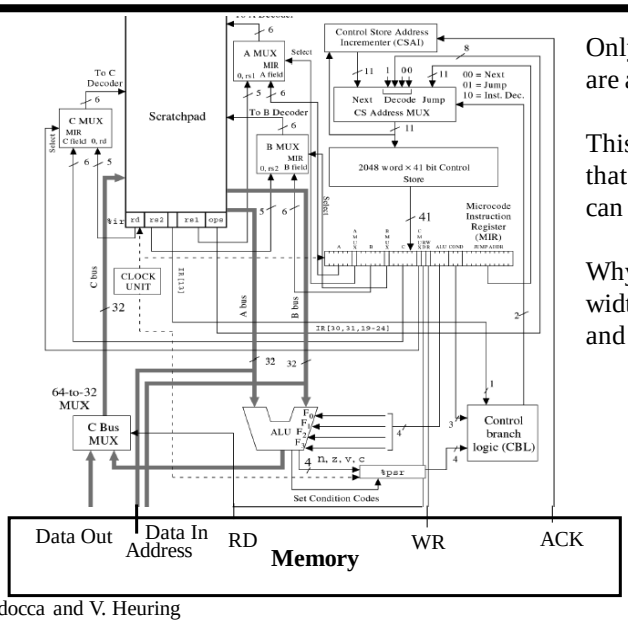
C_2	C_1	C_0	Operation
0	0	0	Use NEXT ADDR
0	0	1	Use JUMP ADDR if n = 1
0	1	0	Use JUMP ADDR if z = 1
0	1	1	Use JUMP ADDR if v = 1
1	0	0	Use JUMP ADDR if c = 1
1	0	1	Use JUMP ADDR if IR[13] = 1
1	1	0	Use JUMP ADDR
1	1	1	DECODE

DECODE Format for Microinstruction Address



- After the instruction is stored in %ir the microprogram of that instruction has to be executed.
- The micro program is somewhere in the control store.
- The start address (11 bits) depends on %ir
  - The top page 168 illustrates that a SETHI instruction (op=00, op2=100) has eight possible start addresses due to random pattern is field IR[21 ..19].
  - To force one unique start address fill the random pattern with zeros (middle page 168).

ARC with memory



Only three multiplexers are added (A Mux,..).

This makes is possible that *rs1* or *microword* can select a register.

Why the difference in bit width of C mux inputs: 5 and 6 bits?

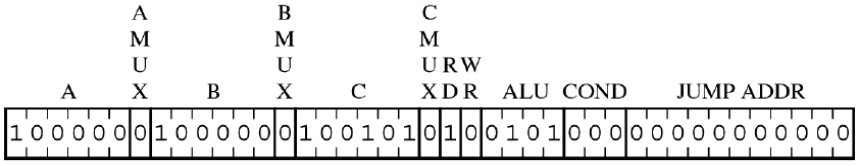
Partial ARC Microprogram (symbolic)

Address	Operation Statements	Comment
0:	R[ir] ← AND(R[pc],R[pc]); READ;	/ Read an ARC instruction from main memory
1:	DECODE;	/ 256-way jump according to opcode
	/ **sethi**	
1152:	R[rd] ← LSHIFT10(ir); GOTO 2047;	/ Copy imm22 field to target register
	/ **call**	
1280:	R[15] ← AND(R[pc],R[pc]);	/ Save %pc in %r15
1281:	R[temp0] ← ADD(R[ir],R[ir]);	/ Shift disp30 field left
1282:	R[temp0] ← ADD(R[temp0],R[temp0]);	/ Shift again
1283:	R[pc] ← ADD(R[pc],R[temp0]);	/ Jump to subroutine
	GOTO 0;	
	/ **addcc**	
1600:	IF R[IR[13]] THEN GOTO 1602;	/ Is second source operand immediate?
1601:	R[rd] ← ADDCC(R[rs1],R[rs2]);	/ Perform ADDCC on register sources
	GOTO 2047;	
1602:	R[temp0] ← SEXT13(R[ir]);	/ Get sign extended simm13 field
1603:	R[rd] ← ADDCC(R[rs1],R[temp0]);	/ Perform ADDCC on register/simm13
	GOTO 2047;	/ sources
2047:	R[pc] ← INCPC(R[pc]); GOTO 0;	/ Increment %pc and start over

Each line is a micro word in the control store.
You should be able to convert a line to a micro word.
Check the start address for instruction *addcc*
Notice the use of bit13 (address 1600) to select register or immediate

Translating the Microprogram

```
0: R[ir] ← AND(R[pc],R[pc]); READ; -- fetch instruction
```



PC is stored in %r32 and %ir is %r37

symbolic

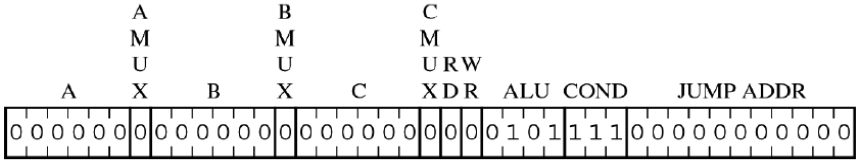
R[pc]	0	R[pc]	0	R[ir]	0	1	0	and	next	-

- is don't care

The ALU operation can be any operation since the A-bus is connected with the the address of the memory. The book uses AND (but the field can also be don't care)

Translating the Microprogram

```
1: DECODE; /256-way jump according to opcode
```



-	-	-	-	R[0]	0	0	0	-	decode	-

- is don't care

Why are **C** and **CMUX** not don't care?

Assembled ARC Microprogram

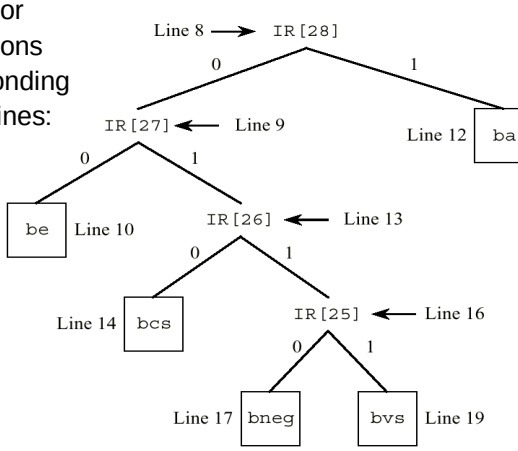
Microstore Address	A	AMUX	B	BMUX	C	CMUX	DRW	DR	ALU	COND	JUMP ADDR
0	1 0 0 0 0 0	0	1 0 0 0 0 0	0	1 0 0 1 0 1	0	1	0	0 1 0 1	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	1 1 1	0 0 0 0 0 0 0 0 0 0 0
1152	1 0 0 1 0 1	0	0 0 0 0 0 0	0	0 0 0 0 0 0	1	0	0	1 0 1 0	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1280	1 0 0 0 0 0	0	1 0 0 0 0 0	0	0 0 1 1 1 1	0	0	0	0 1 0 1	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1281	1 0 0 1 0 1	0	1 0 0 1 0 1	0	1 0 0 0 0 1	0	0	0	1 0 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1282	1 0 0 0 0 1	0	1 0 0 0 0 1	0	1 0 0 0 0 1	0	0	0	1 0 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1283	1 0 0 0 0 0	0	1 0 0 0 0 1	0	1 0 0 0 0 0	0	0	0	1 0 0 0	1 1 0	0 0 0 0 0 0 0 0 0 0 0
1600	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	1 0 1	1 1 0 0 1 0 0 0 0 1 0
1601	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0	0	0 0 1 1	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1602	1 0 0 1 0 1	0	0 0 0 0 0 0	0	1 0 0 0 0 1	0	0	0	1 1 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1603	0 0 0 0 0 0	1	1 0 0 0 0 1	0	0 0 0 0 0 0	1	0	0	0 0 1 1	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1604	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	1 0 1	1 1 0 0 1 0 0 0 1 1 0
1605	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0	0	0 0 0 0	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1606	1 0 0 1 0 1	0	0 0 0 0 0 0	0	1 0 0 0 0 1	0	0	0	1 0 1 1	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1607	0 0 0 0 0 0	1	1 0 0 0 0 1	0	0 0 0 0 0 0	1	0	0	0 0 0 0	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1608	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	1 0 1	1 1 0 0 1 0 0 1 0 1 0
1609	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0	0	0 0 0 1	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1610	1 0 0 1 0 1	0	0 0 0 0 0 0	0	1 0 0 0 0 1	0	0	0	1 0 1 1	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1611	0 0 0 0 0 0	1	1 0 0 0 0 1	0	0 0 0 0 0 0	1	0	0	0 0 0 1	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1624	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	1 0 1	1 1 0 0 1 0 1 1 0 1 0
1625	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0	0	0 0 1 0	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1626	1 0 0 1 0 1	0	0 0 0 0 0 0	0	1 0 0 0 0 1	0	0	0	1 0 1 1	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1627	0 0 0 0 0 0	1	1 0 0 0 0 1	0	0 0 0 0 0 0	1	0	0	0 0 1 0	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1688	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	0 0 0	1 1 0 1 0 0 1 1 0 1 0
1689	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0 0 0 0 0 0	1	0	0	0 1 0 0	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1690	1 0 0 1 0 1	0	0 0 0 0 0 0	0	1 0 0 0 0 1	0	0	0	1 0 1 1	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1691	0 0 0 0 0 0	1	1 0 0 0 0 1	0	0 0 0 0 0 0	1	0	0	0 1 0 0	1 1 0	1 1 1 1 1 1 1 1 1 1 1
1760	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0 0 0 0 0 0	0	0	0	0 1 0 1	1 0 1	1 1 0 1 1 0 0 0 1 0
1761	0 0 0 0 0 0	1	0 0 0 0 0 0	1	1 0 0 0 0 0	0	0	0	1 0 0 0	1 1 0	0 0 0 0 0 0 0 0 0 0 0
1762	1 0 0 1 0 1	0	0 0 0 0 0 0	0	1 0 0 0 0 1	0	0	0	1 1 0 0	0 0 0	0 0 0 0 0 0 0 0 0 0 0
1763	0 0 0 0 0 0	1	1 0 0 0 0 1	0	1 0 0 0 0 0	0	0	0	1 0 0 0	1 1 0	0 0 0 0 0 0 0 0 0 0 0
1792	0 0 0 0 0 0	1	0 0 0 0 0 0	1	1 0 0 0 0 1	0	0	0	1 0 0 0	1 0 1	1 1 1 0 0 0 0 0 1 0

Other instructions

- **Other instructions .. Try it yourself!**
- ***Branch* instructions are implemented a little bit tricky. Many jumps (=clock cycles ➔ delay!) are made in the control store.**

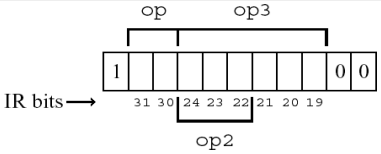
Branch Decoding

Decoding tree for branch instructions shows corresponding microprogram lines:



cond				branch
28	27	26	25	
0	0	0	1	be
0	1	0	1	bcs
0	1	1	0	bneg
0	1	1	1	bvs
1	0	0	0	ba

Start address of instruction in control store



- **The distance between two start addresses of an instruction is 4.**
- **As a consequence there are only 4 consecutive addresses available for the micro program.**
- **If an instruction needs more space the addresses from 2 until 1023_{10} (0111111111_2) can be used.**

Example: Add the `subcc` Instruction

Add instruction **`subcc`** (subtract) to the ARC instruction set. **`subcc`** uses the Arithmetic format and op3 = 001100. What is the start address in control store?

```
1584:   R[temp0] ←SEXT13( R[ir] ) ;        / Extract rs2 operand
        IF IR[13] THEN GOTO 1586;          / Is second source immediate?
1585:   R[temp0] ←R[rs2];                  / second operand in register file
1586:   R[temp0] ←ORN (R[0], R[temp0]);    / one's complement of R[temp0]
1587:   R[temp0] ←INC(R[temp0]);           / add 1 (two's complement)
        GOTO 1603                          / goto "addition (see page 165)
```

	A	AMUX	B	BMUX	C	CMUX	DR	WR	ALU	COND	JUMP ADDR
1584	R[ir]	0	-	-	R[temp0]	0	0	0	sext	ir[13]	1586
1585	R[0]	0	-	1	R[temp0]	0	0	0	add	next	-
1586	R[0]	0	R[temp0]	0	R[temp0]	0	0	0	orn	next	-
1587	R[temp0]	0	-	-	R[temp0]	0	0	0	inc	jump	1603

Example: Add the `subcc` Instruction

Add instruction **`subcc`** (subtract) to the ARC instruction set. **`subcc`** uses the Arithmetic format and op3 = 001100. What is the start address in control store?

```
1584:   R[temp0] ←SEXT13( R[ir] ) ;        / Extract rs2 operand
        IF IR[13] THEN GOTO 1586;          / Is second source immediate?
1585:   R[temp0] ←R[rs2];                  / second operand in register file
1586:   R[temp0] ←ORN (R[0], R[temp0]);    / one's complement of R[temp0]
1587:   R[temp0] ←INC(R[temp0]);           / add 1 (two's complement)
        GOTO 1603                          / goto "addition (see page 165)
```

	A	AMUX	B	BMUX	C	CMUX	DR	WR	ALU	COND	JUMP ADDR
1584	10010	1	00000000	1	00000100	0	0	0	1100	101	11000110010
1585	00000	0	00000000	1	00000100	0	0	0	1000	000	00000000000000
1586	00001	0	00000000	1	00000100	0	0	0	0111	000	00000000000000
1587	00001	0	00000000	1	00000100	0	0	0	1101	110	11001000011

© 2007 M. Murdocca and V. Heuring

What next?

- Make the exercises (and Thursday afternoon)!
- Friday at 8.45 Q&A session
- Friday at 13.45 Q&A session
- Monday a diagnostic test:
  - Closed book
  - Simple calculator is allowed
  - Take with you a copy of the “documentation_ARC.pdf” (4 pages) (at the exam you get it from us)