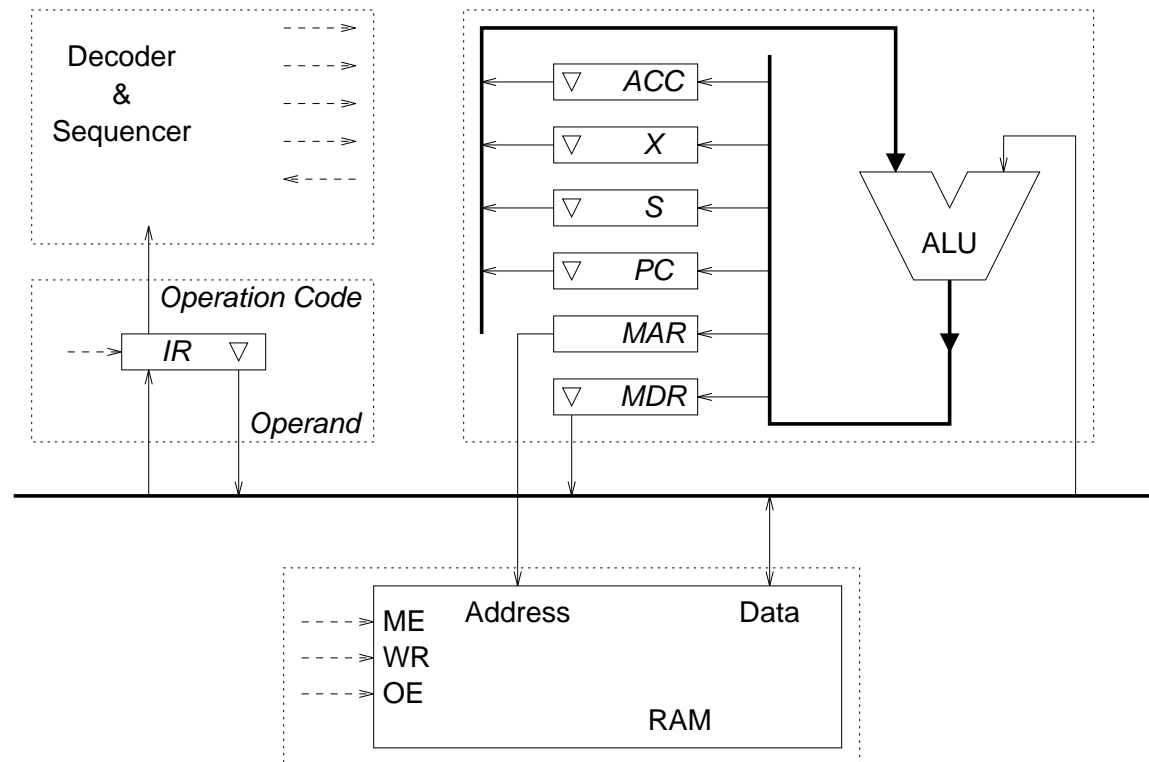
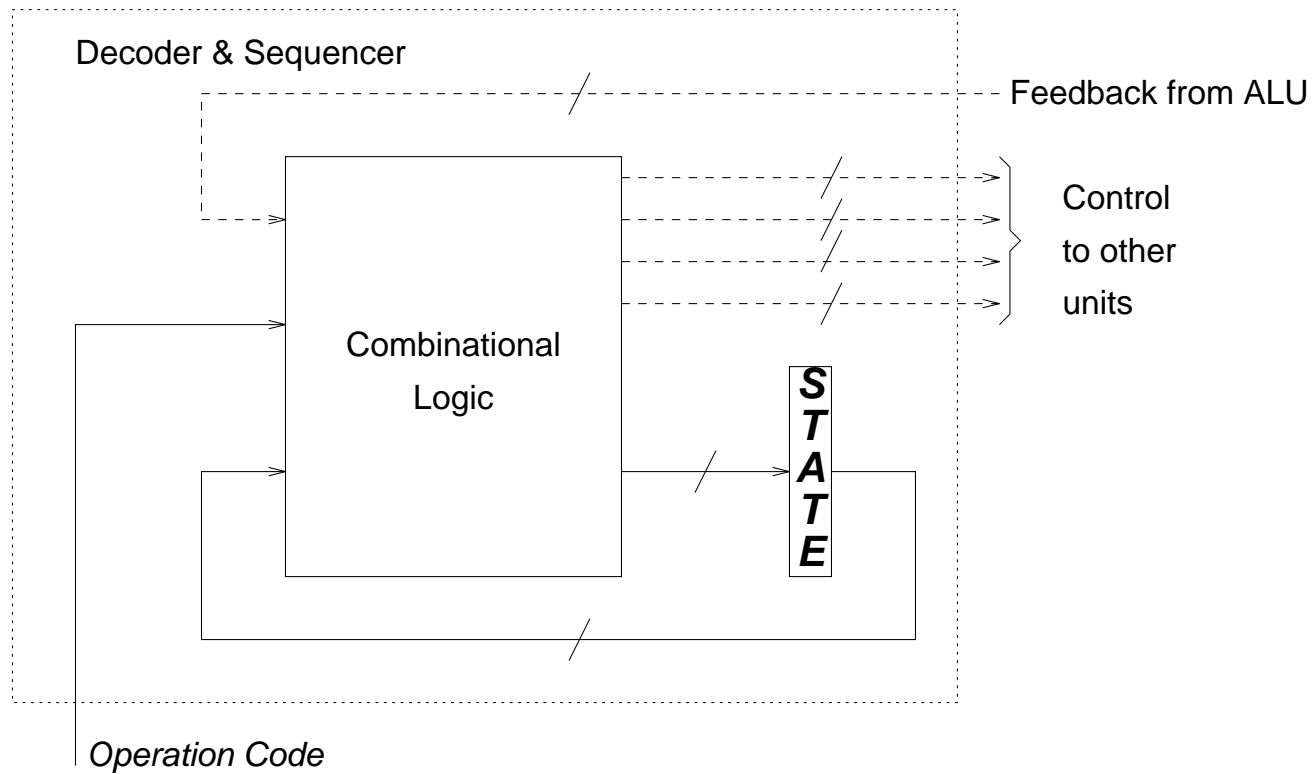


Control Unit Design



Control Unit Design



Control Unit Design

- Control
 - Memory Bus
Enable_MDR, Enable_Op, OE, Load_IR, WR.
 - Register O/P Bus
Enable_ACC, Enable_X, Enable_S, Enable_PC.
 - Register I/P Bus
Load_ACC, Load_X, Load_S, Load_PC, Load_MAR, Load_MDR.
 - ALU
F0-F5 (ALU_function), Multiword, Plus_1, Arithmetic_Shift.
Update_C, Update_V, Update_N, Update_Z.
- Feedback
 - Condition Codes
C, V, N, Z.

Instruction Set Summary

- Arithmetic Instructions

	Inherent #3 +w	Immediate #3 +w	Direct #4 +2w	Indirect #5 +3w	Indexed #4 +2w	Indexed Indirect #5 +3w	Stack (pull) #5 +2w
CLR	●						
INC	●						
DEC	●						
NEG	●						
COM	●						
LSL	●						
LSR	●						
ASR	●						
ROL	●						
ROR	●						
ADD		●	●	●	●	●	●
SUB		●	●	●	●	●	●
ADC		●	●	●	●	●	●
SBC		●	●	●	●	●	●
AND		●	●	●	●	●	●
OR		●	●	●	●	●	●
EOR		●	●	●	●	●	●

Instruction Set Summary

- Load Register from Memory

	Immediate $\#3 + w$	Direct $\#4 + 2w$	Indirect $\#5 + 3w$	Indexed $\#4 + 2w$	Indexed Indirect $\#5 + 3w$	Stack (pull) $\#5 + 2w$
LDA	●	●	●	●	●	●
LDX	●	●	●			●
LDS	●	●	●	●	●	

- Store Register to Memory

	Direct $\#4 + 2w$	Indirect $\#5 + 3w$	Indexed $\#4 + 2w$	Indexed Indirect $\#5 + 3w$	Stack (push) $\#4 + 2w$
STA	●	●	●	●	●
STX	●	●			●
STS	●	●	●	●	

Instruction Set Summary

- Control Transfer Instructions¹

	PC Relative		Direct		Indirect		Stack (pull)
	#3 +w	#5+2w	#3 +w	#5+2w	#4 +2w	#6 +3w	#5 +2w
BRA	●						
BRN	●						
BCC	●						
BCS	●						
BVC	●						
BVS	●						
BEQ	●						
BNE	●						
BLT	●						
BGE	●						
BGT	●						
BLE	●						
BSR		●					
JMP			●		●		
JSR				●		●	
RTS							●

¹note that the BSR & JSR are identical to the BRA & JMP instructions except that BSR & JSR store the old PC value on the stack for later retrieval by RTS.

Control Unit Design

- Complexity

Consider the complexity for an implementation using synchronous memory²:

- 28 Control signals
- 4 Feedback signals
- 8-bit State (201 States)
- 7-bit Opcode (97 Operation Codes)
 - - 39 Instructions
 - - 9 Addressing Modes

Combinational logic has 19 (4+7+8) inputs and 36 (28+8) outputs.

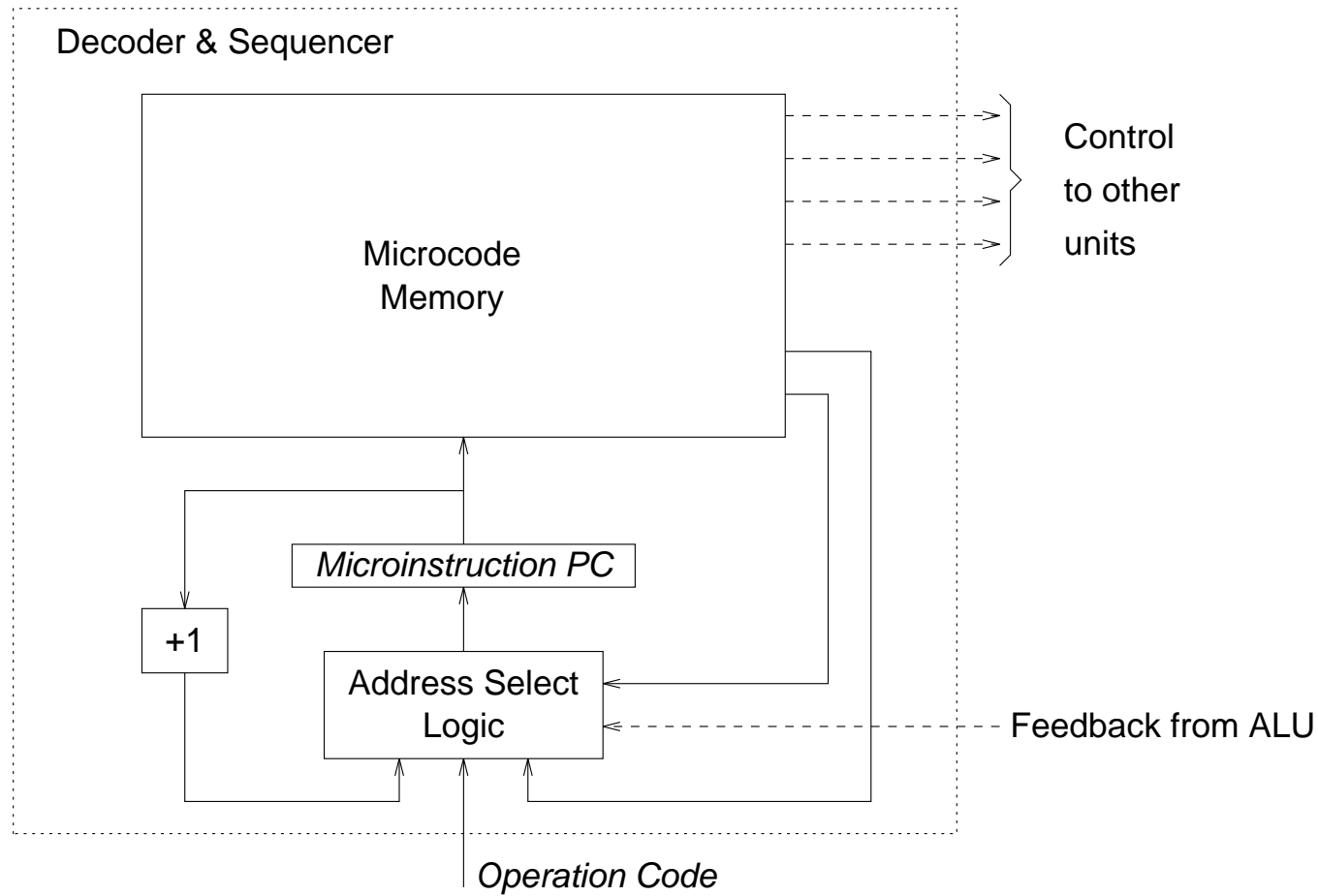
² $ME = \overline{CLK}$ hence no wait states

Control Unit Design

- Options
 - Hardwired Logic
 - Computer assisted formal methods for random logic
 - Large & complex
 - PLA
 - More compact
 - Size limited – even this problem is too big for a single PLA
 - Microcode
 - Simple implementation of complex control task
 - Control is reduced to a software problem

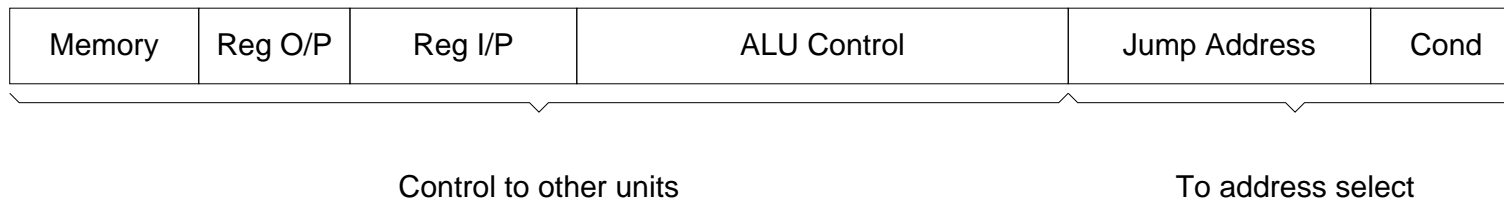
Control Unit Design

- Microcode



Control Unit Design

- Microcode Instructions



- a new microinstruction is executed every cycle
- each instruction contains all the control signals required during a particular clock cycle
- the signals are usually arranged into fields according to function

Control Unit Design

- Microcode Execution

Operation code maybe simply a microcode address.

In each cycle the Microinstruction Program Counter may be loaded with one of the following:

- $\mu\text{PC} + 1$
execute next microinstruction (default)
- Opcode
begin execution of next macroinstruction after instruction fetch
- Jump Address
jump to another section of microcode – may be dependent on ALU feedback

Control Unit Design

- Field Encoding

Even for this simple example we have a very wide microinstruction. To reduce the size of the microcode memory it is usual to encode the microinstruction fields.

Since only one register may drive the Register O/P bus we can encode the field from four bits down to two³:

Encoded Field		
0	0	Enable_ACC
0	1	Enable_X
1	0	Enable_S
1	1	Enable_PC

³although the microcode memory is now smaller we will need some form of additional decoding

Control Unit Design

- Two Level Microcode

Many microinstructions will have the same control signals. With a two level microcode/nanocode system we can merge such instructions.

- Microcode

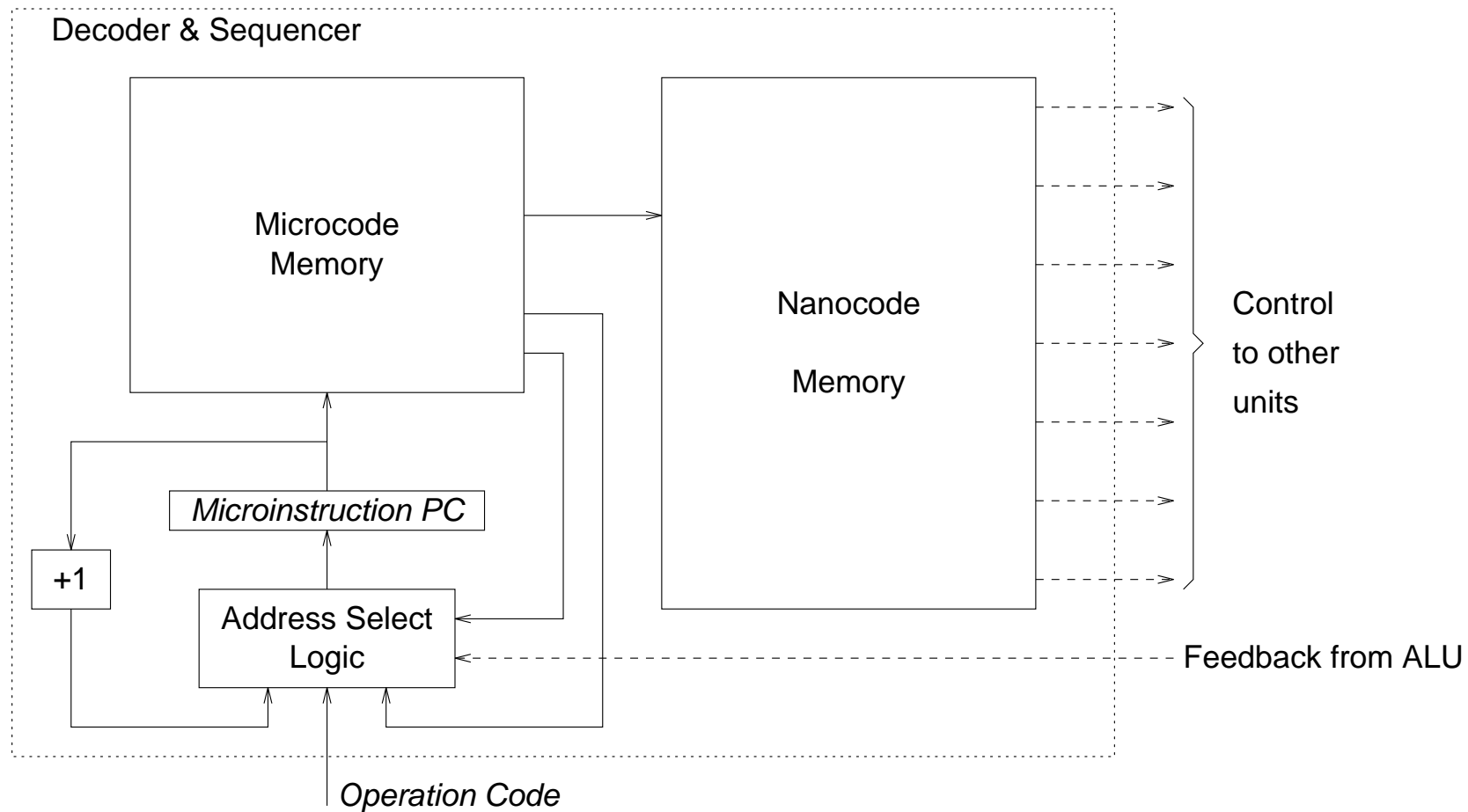
- - large number of narrow microinstructions
 - - controls sequence of microinstructions
 - - controls selection of nanoinstructions

- Nanocode

- - small number of wide nanoinstructions
 - - provides control to other units

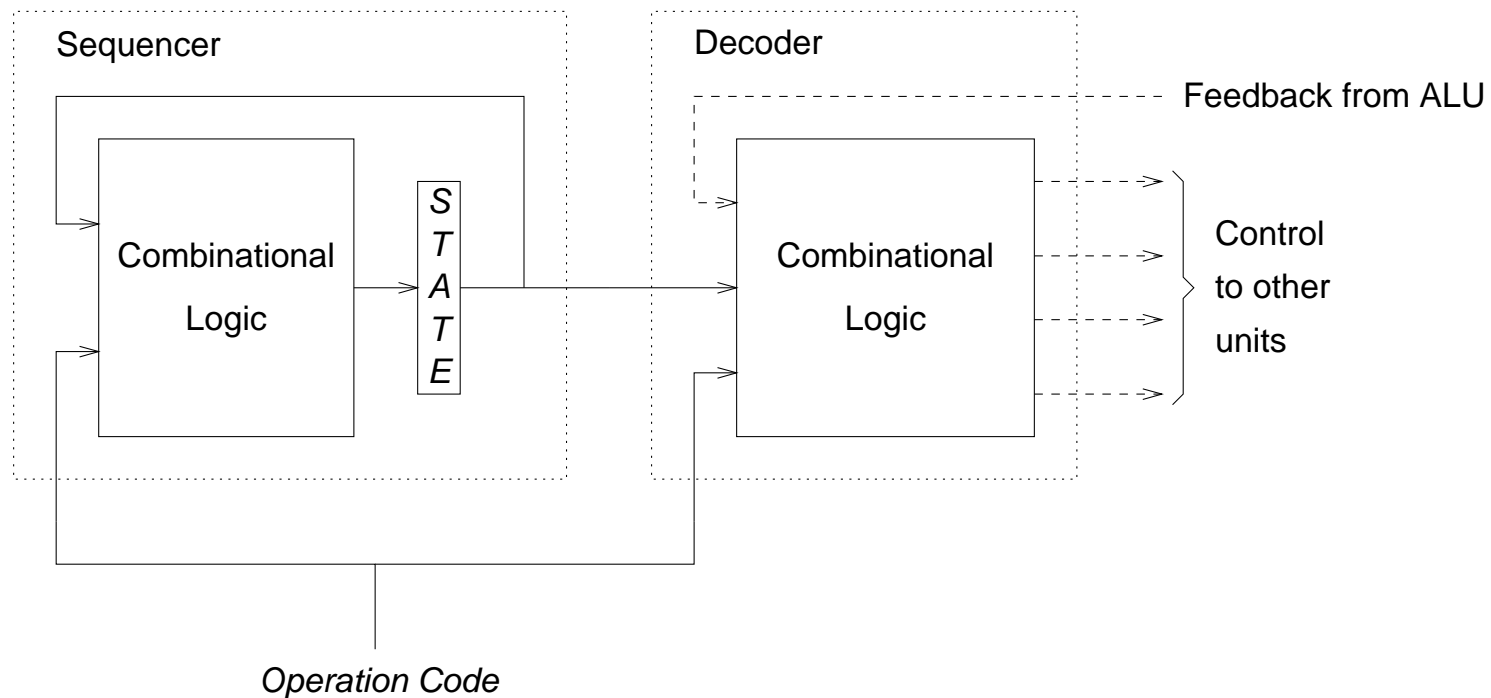
Control Unit Design

- Nanocode System



Control Unit Design

To simplify the control unit for use in high clock rate modern microprocessors we can first separate the tasks of sequencing and decoding:

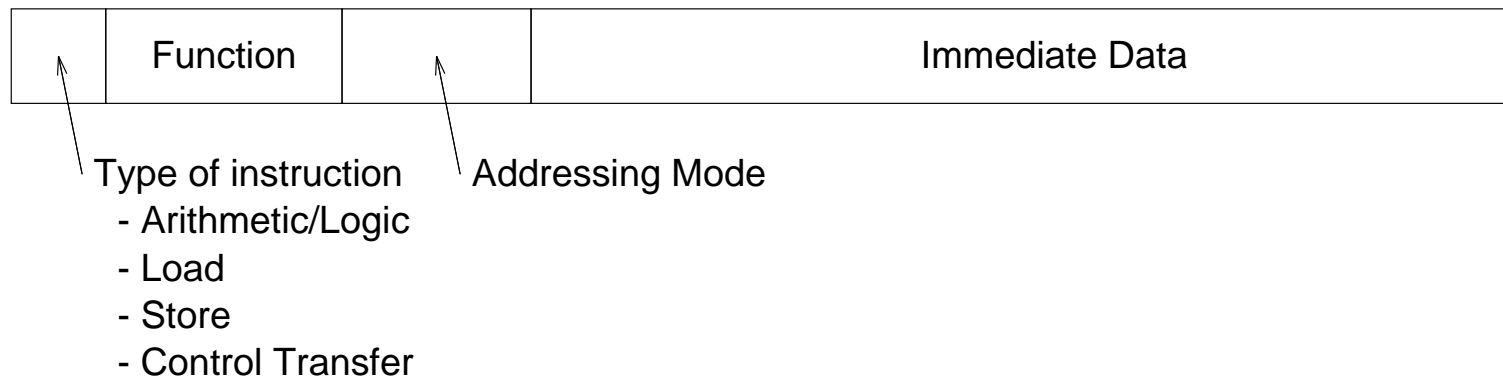


Here the sequencer merely keeps track of instruction execution.

Control Unit Design

- The majority of the controller state is in the instruction register.
- Careful choice of opcodes can significantly reduce decoder complexity⁴.

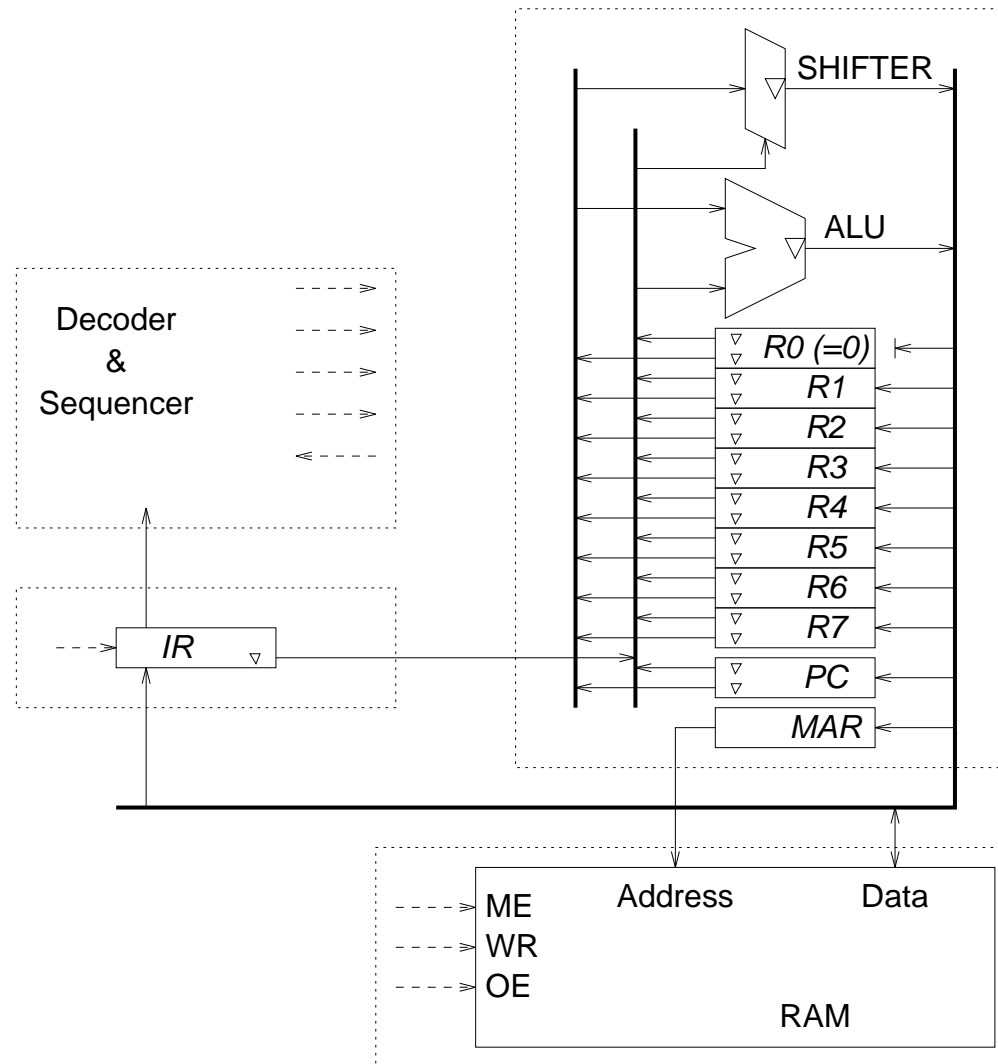
Macroinstruction fields



Division of the instruction into fields will simplify the decoder and sequencer at the expense of a less efficient instruction coding. The best results are obtained for RISC style architectures.

⁴choice of opcodes is now a problem in state allocation.

RISC



RISC

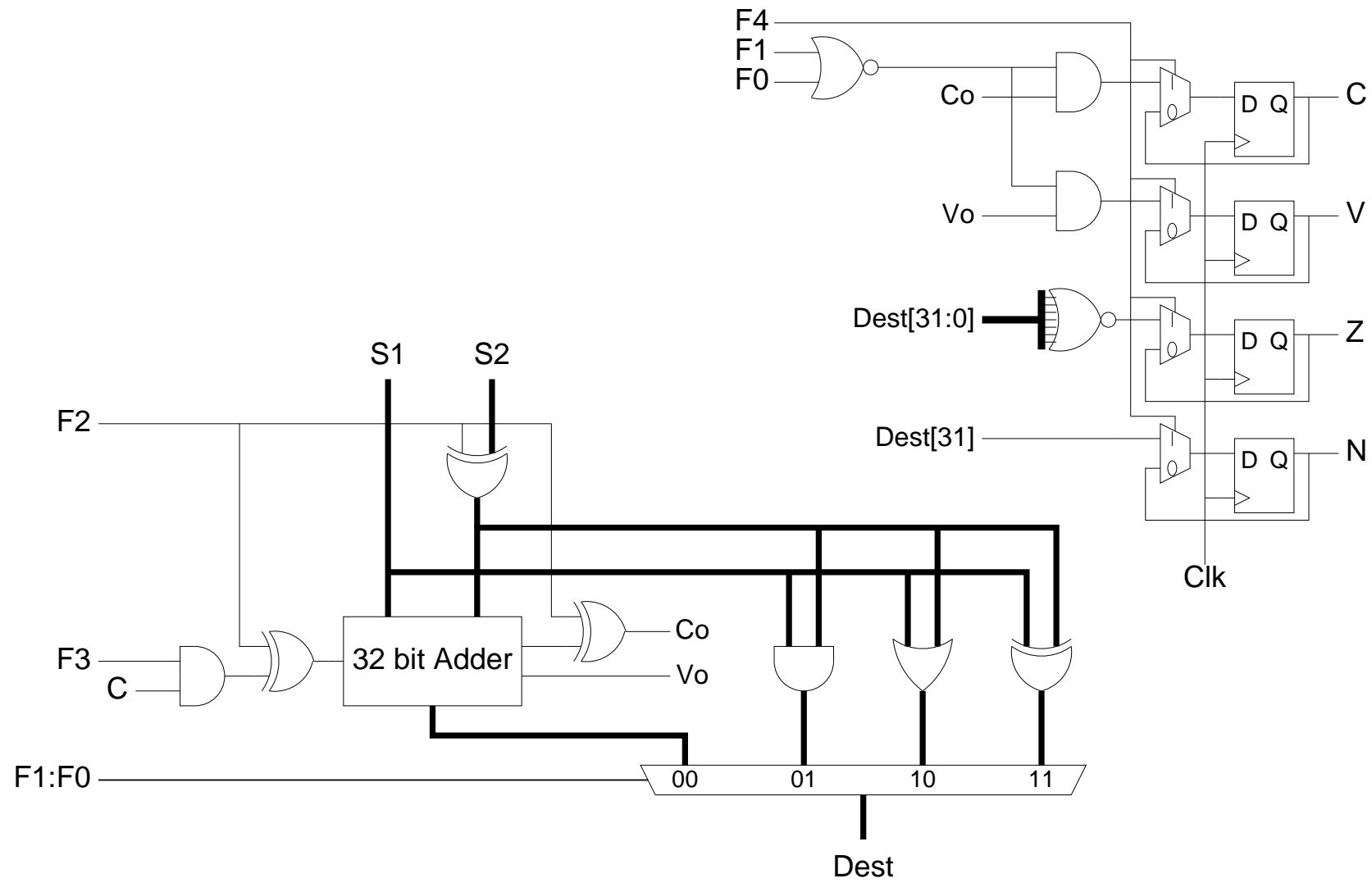
Typical RISC

- Register register architecture
- Few instructions
- Very few addressing modes

The processor described here is based on a subset of the SPARC specification⁵.

⁵note that SPARC is not an architecture only a specification based around a complete instruction set

RISC



RISC

The ALU supports a small number of functions based on only five control signals.

A separate barrel shifter supports three simple shift instructions.

ALU instructions

F4:F3:F2 \ F1:F0				
	0 0	0 1	1 0	1 1
0 0 0	ADD	AND	OR	XOR
0 0 1	SUB	ANDN	ORN	XNOR
0 1 0	ADDX			
0 1 1	SUBX			
1 0 0	ADDcc	ANDcc	ORcc	XORcc
1 0 1	SUBcc	ANDNcc	ORNcc	XNORcc
1 1 0	ADDXcc			
1 1 1	SUBXcc			

Shift instructions

SLL SRL SRA

RISC

- All ALU and shift instructions take two operands (either two registers or one register and one immediate) and return a single register result.

SUBX R3, R7, R5 $R5' \leftarrow R3 - R7 - C$

ADD R4, 5, R1 $R1' \leftarrow R4 + 5$

- for special cases we have a pseudo register, $R0$, which always contains zero.

SUB R0, R2, R2 $R2' \leftarrow -R2$

- we may even execute an instruction merely for its side effects

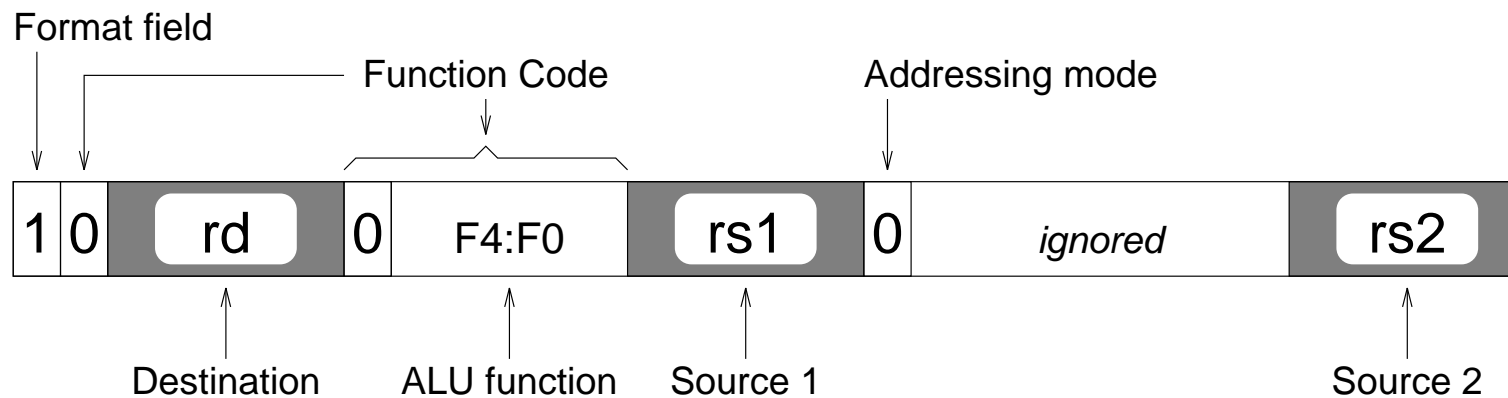
SUBCC R5, 201, R0

RISC

Instruction Format

- Arithmetic and logic instructions contain all the relevant fields for register selection and ALU control.

The decoder is responsible for enabling these fields during the correct cycle.

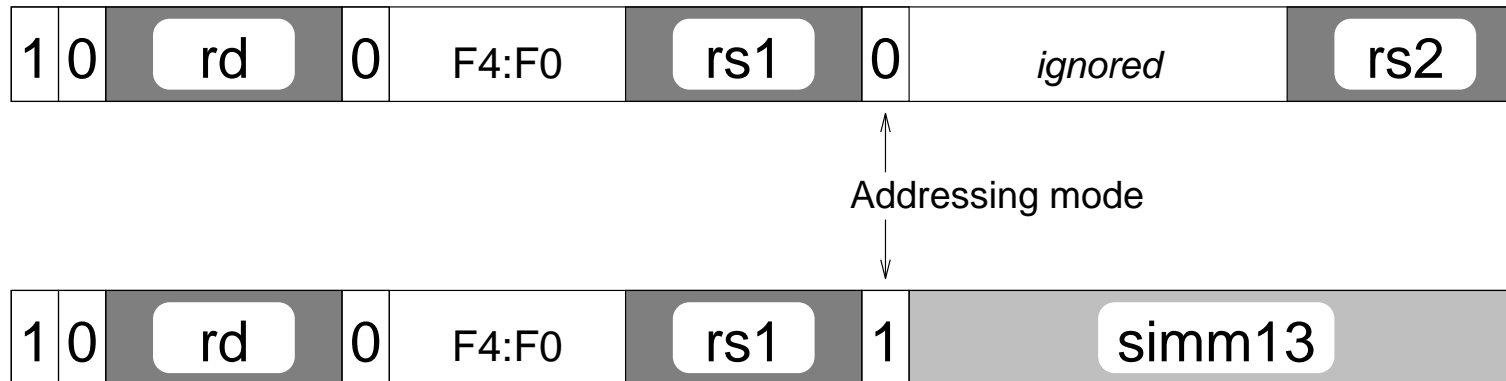


RISC

Variable Instruction Format

The use of variable fields allows us to specify more information within a limited instruction width⁶.

For simplicity the signed immediate may only be used on the **s2** bus.

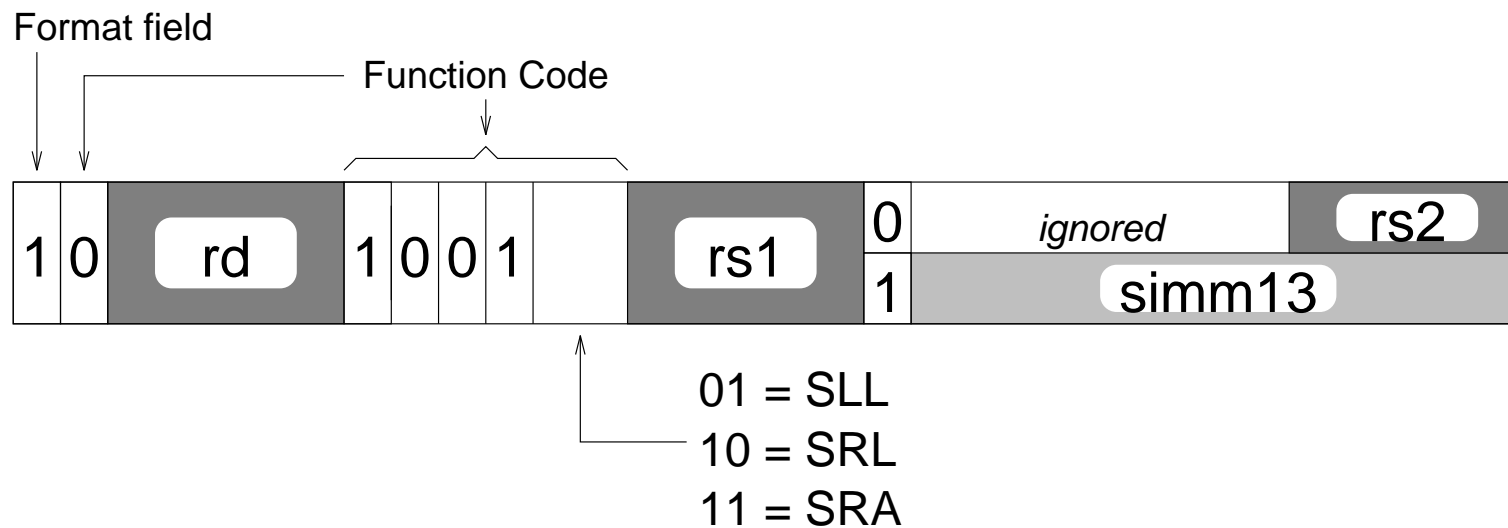


⁶note that the **rs2** field is not always present but when present it is always in the same place.

RISC

- Shift instructions are indicated by a different function code.

The decoder will use a part of the function code in order to select which functional unit drives the destination bus.



RISC

- Load and store instructions use similar instruction formats.

LD [R3+R7], R5

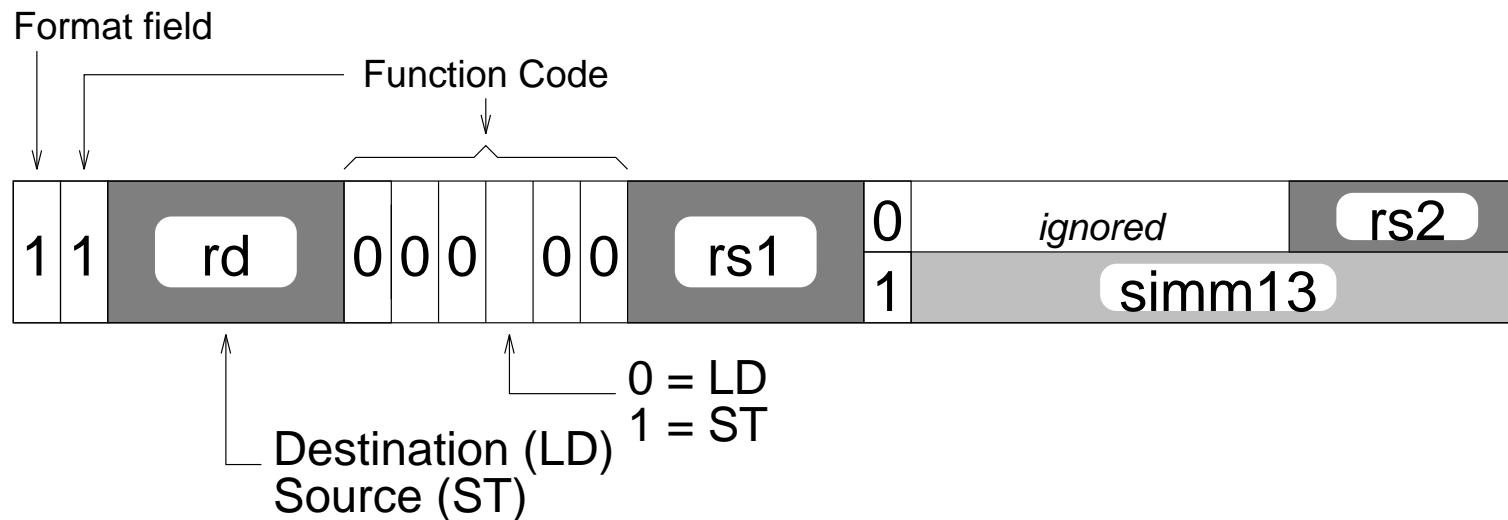
$MAR' \leftarrow R3 + R7$

$R5' \leftarrow mem(MAR)$

ST R1, [R4+5]

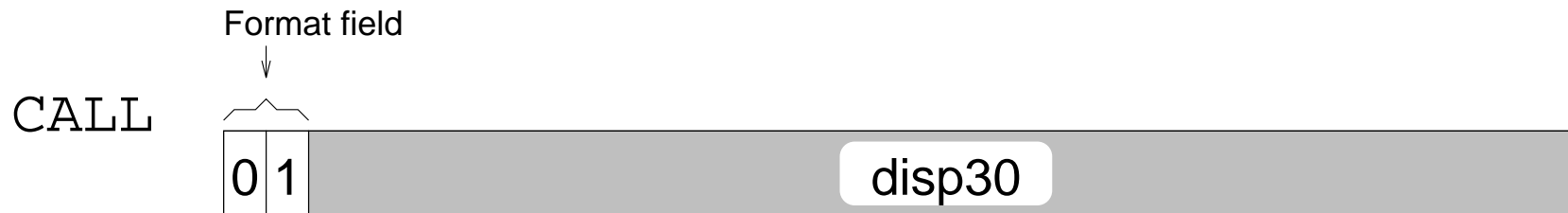
$MAR' \leftarrow R4 + 5$

$mem(MAR)' \leftarrow R1$



RISC

- Control Transfer instructions.



Since instructions are word aligned, any location within the 4 GByte address space may be accessed via the `CALL` instruction⁷.

`CALL label`

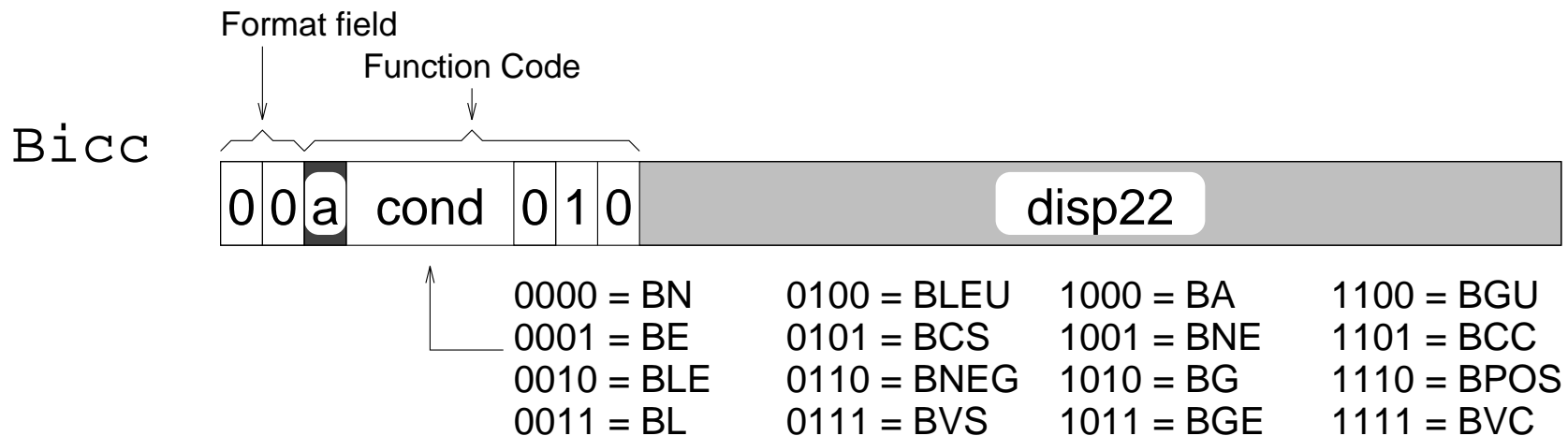
$$R15' \leftarrow PC$$

$$PC' \leftarrow PC + (disp30 * 4)$$

⁷A price is paid for this facility – R15 is fixed as the register storing the return address

RISC

- Control Transfer instructions.



The conditional branch instructions have a shorter range since they need eight bits to specify the branch condition and the annul⁸ status.

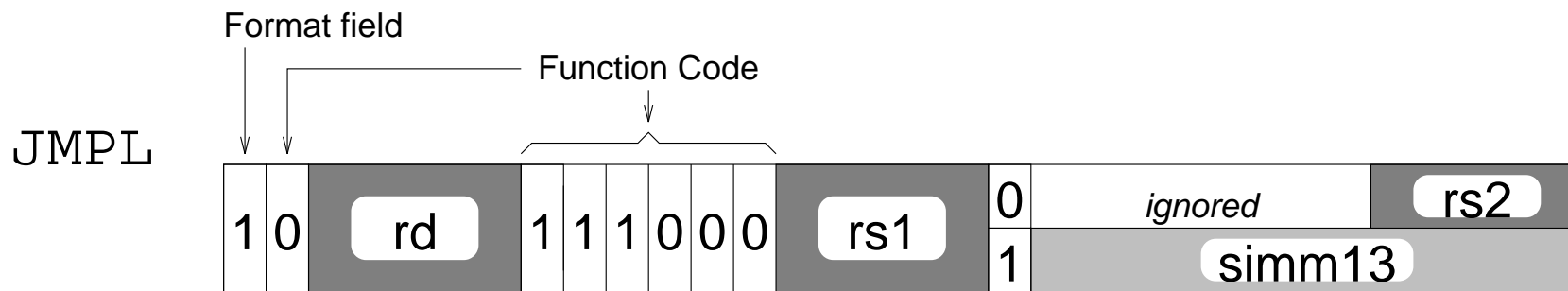
BNE *label*

If ($Z = 0$) then $PC' \leftarrow PC + (disp22 * 4)$

⁸not documented here

RISC

- Control Transfer instructions.



This instruction allows a register indirect jump. Since it saves the old program counter it may be used to jump to or return from a subroutine.

$\text{JMPL } rs1+rs2, rd \quad rd' \leftarrow PC$
 $PC' \leftarrow rs1 + rs2$

The documentation given here for control transfer instructions has omitted any mention of the pipelining and register window features of the SPARC specification. The resulting description is self consistent but inaccurate.

RISC

- Using an `ADD` instruction we can set the value of a register in the range -4096 to $+4095$ ⁹.

`ADD R0, simm13, rd` $rd' \leftarrow simm13$

- The `SETHI` instruction enables the setting of the more significant register bits.



`SETHI imm22, rd` $rd < 31 : 10 >' \leftarrow imm22$
 $rd < 9 : 0 >' \leftarrow 0$

⁹note that the signed immediate is sign extended before placement on the `s2` bus

RISC

- Manipulation of immediate and displacement values:

