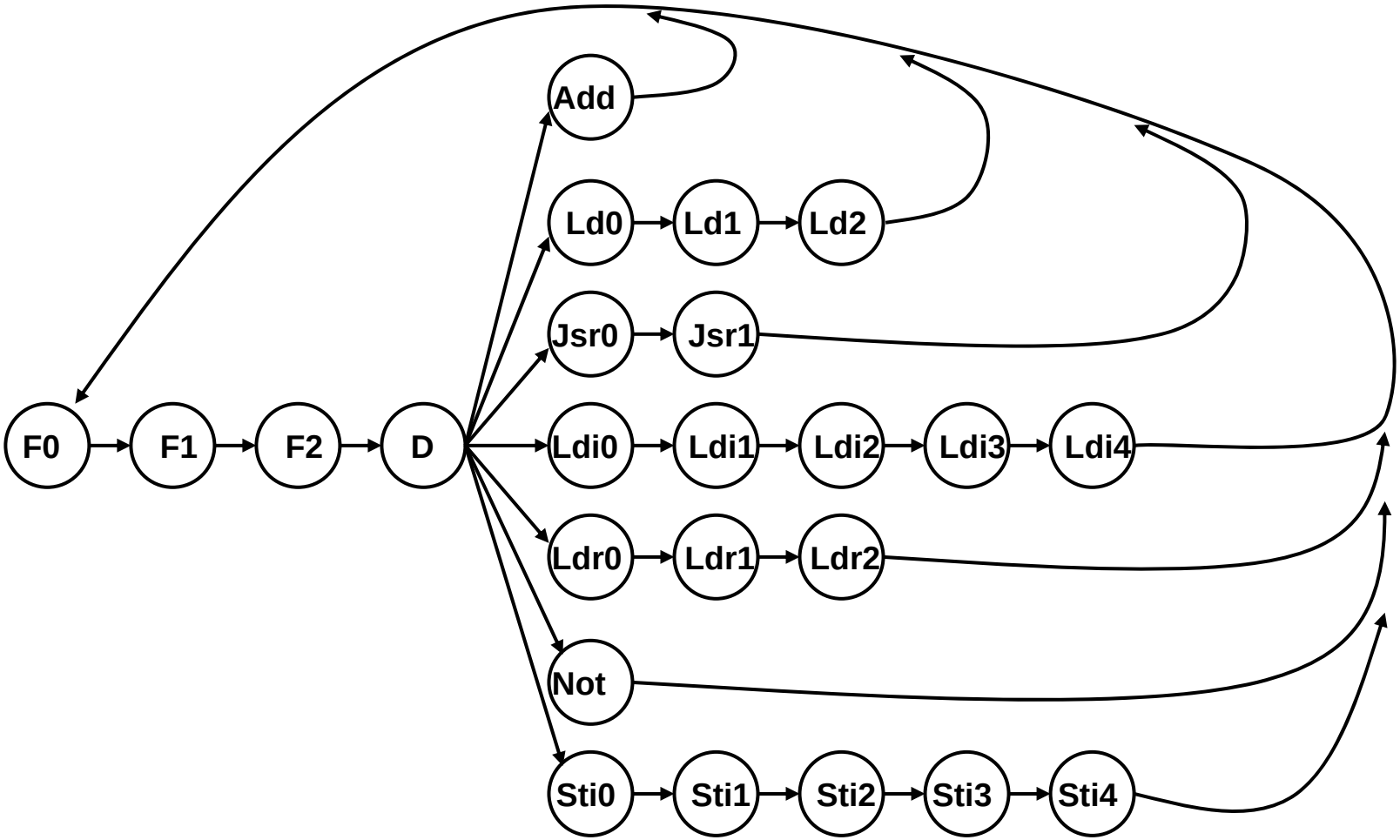
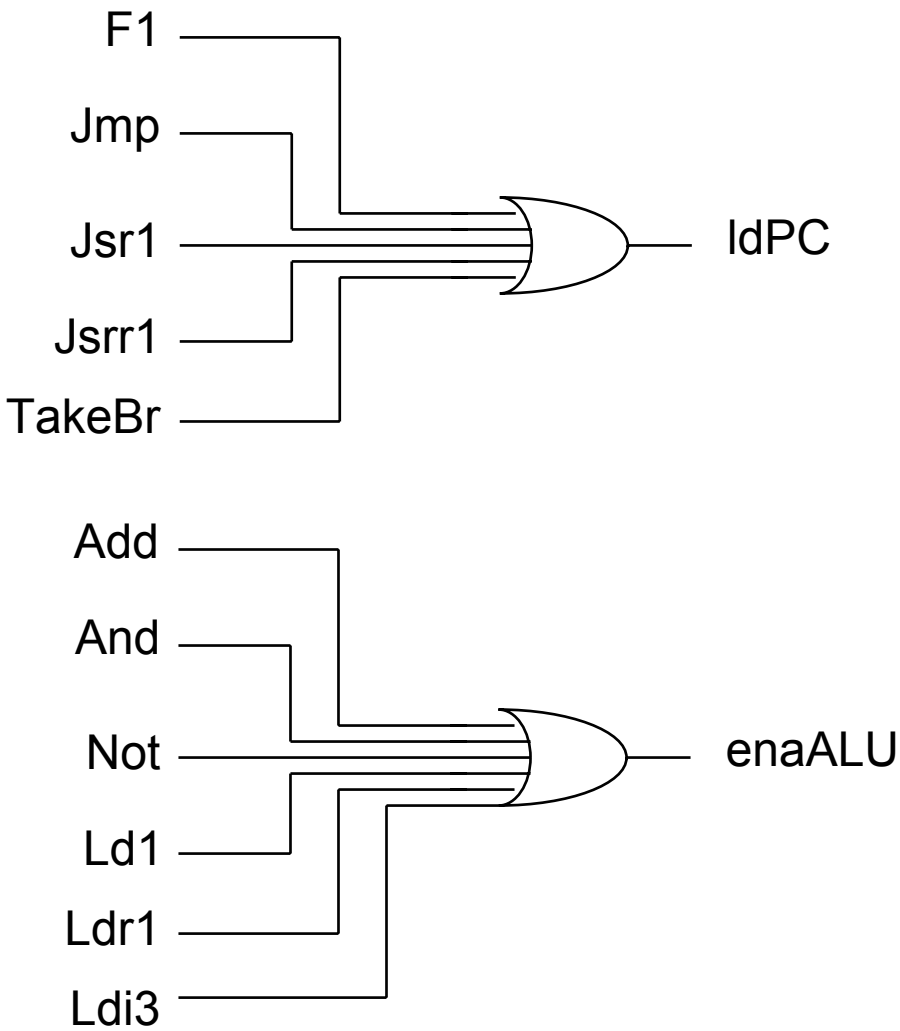
Microprogrammed Control

Sample Control Unit FSM



Sample Output Forming Logic

One-hot encoding leads to pretty simple OFL!



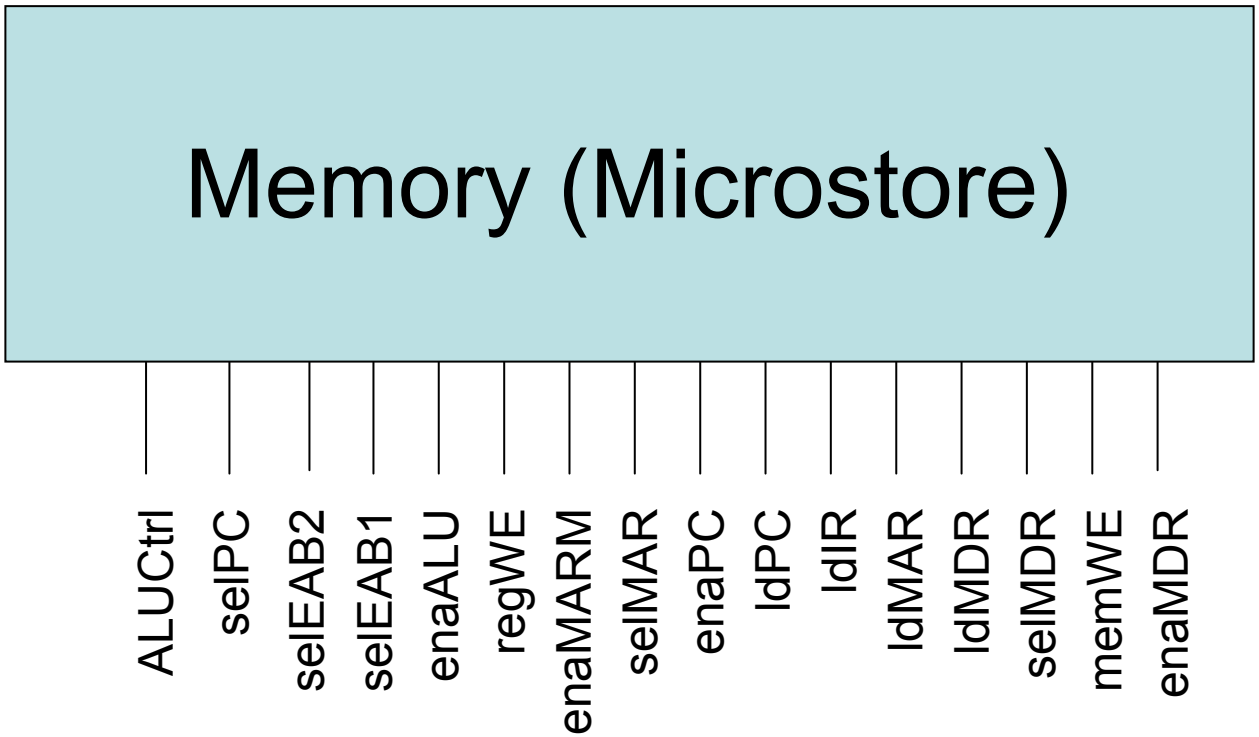
The Control Unit

- The LC-3 control lecture discussed a hard-wired approach to controller design
  - The functionality is fixed after the design
- Microprogrammed controllers allow functionality to be easily changed
  - Instructions can be added or changed with only minor changes to the *microcode*

Microprogrammed Control

An internal memory called the **control store** (or Microstore) contains the settings for all of the control signals for all.

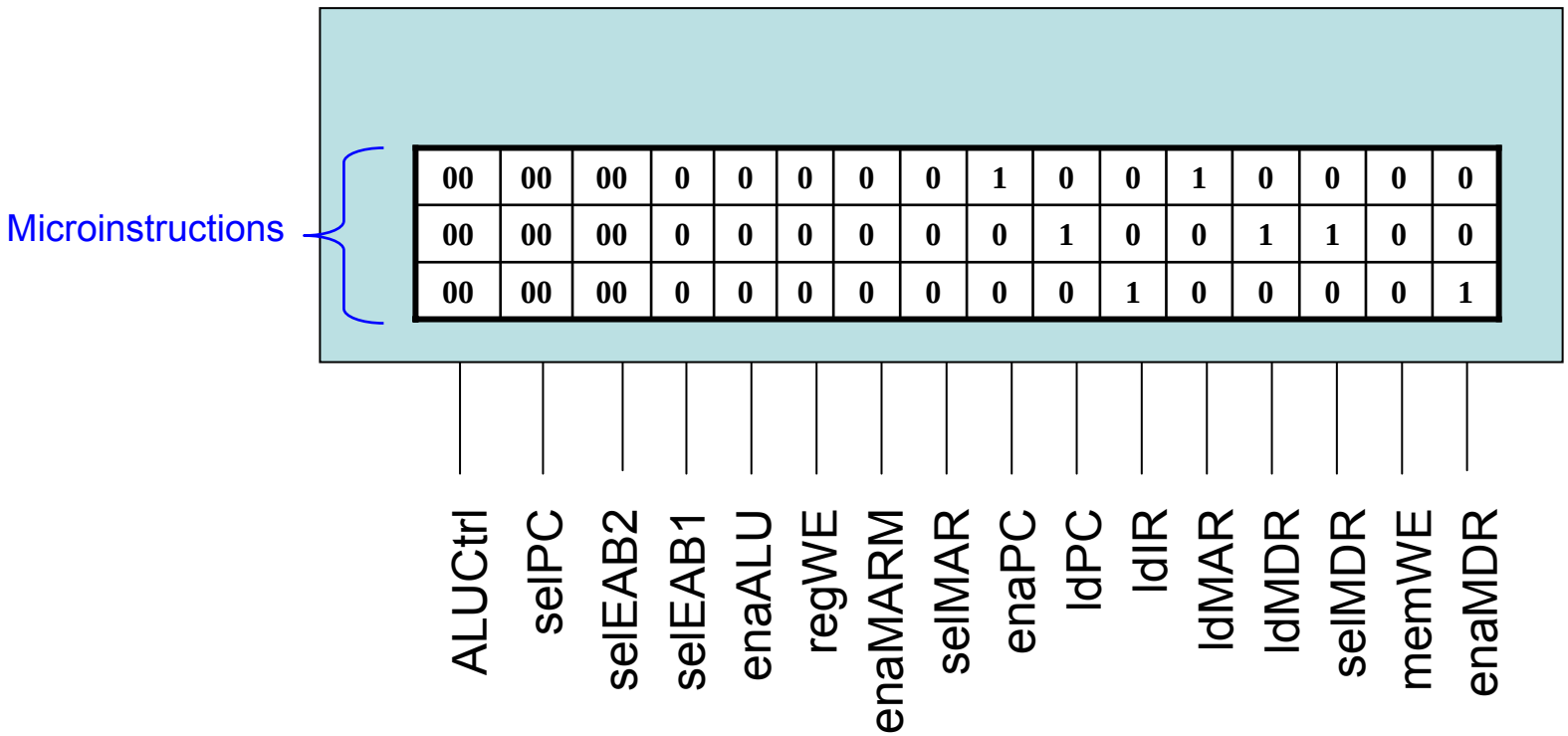
Each memory location contains one complete set of control values.



Microprogrammed Control

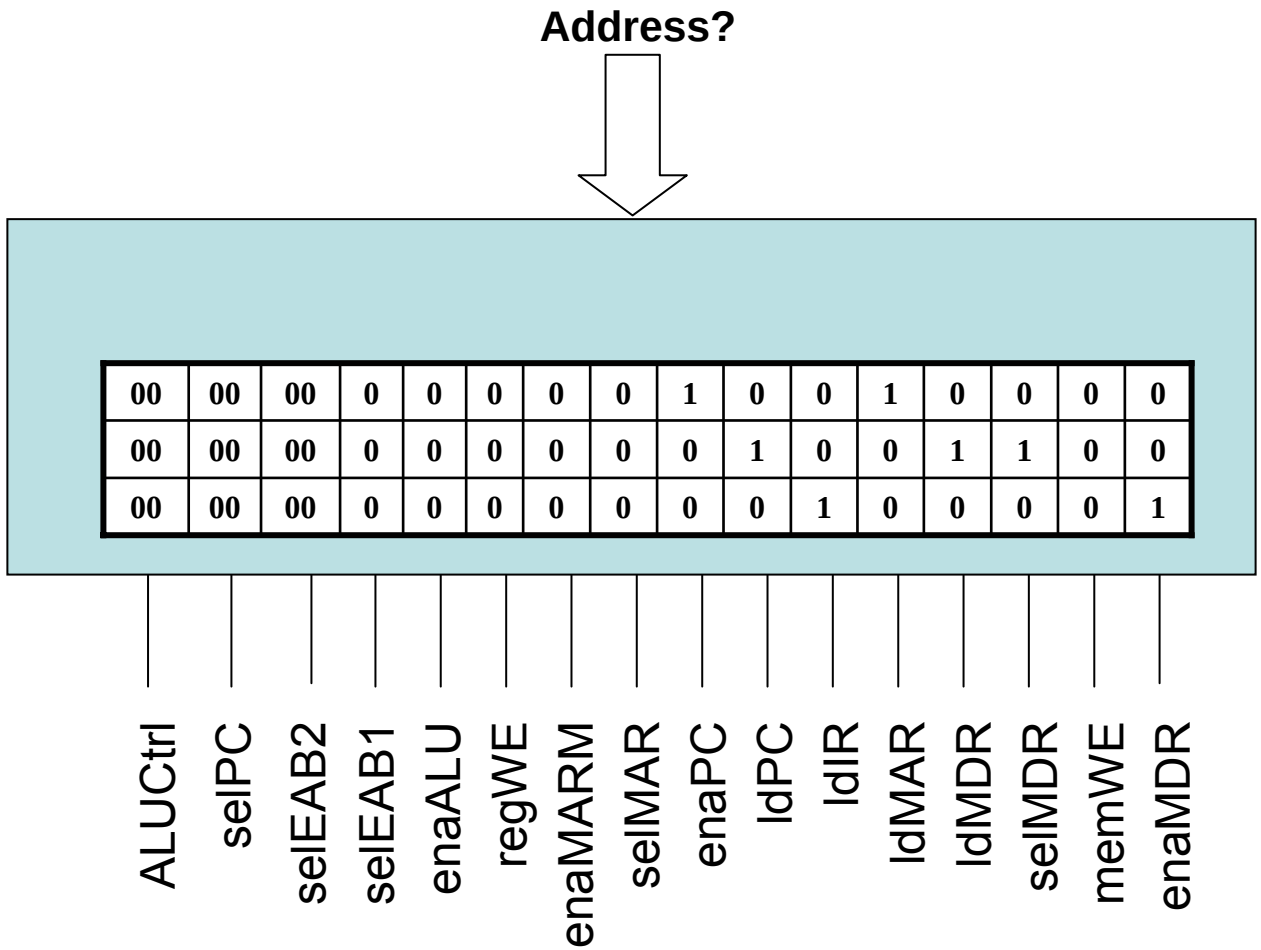
An internal memory called the **control store** (or microstore) contains the settings for all of the control signals for all.

Each memory location contains one complete set of control values.



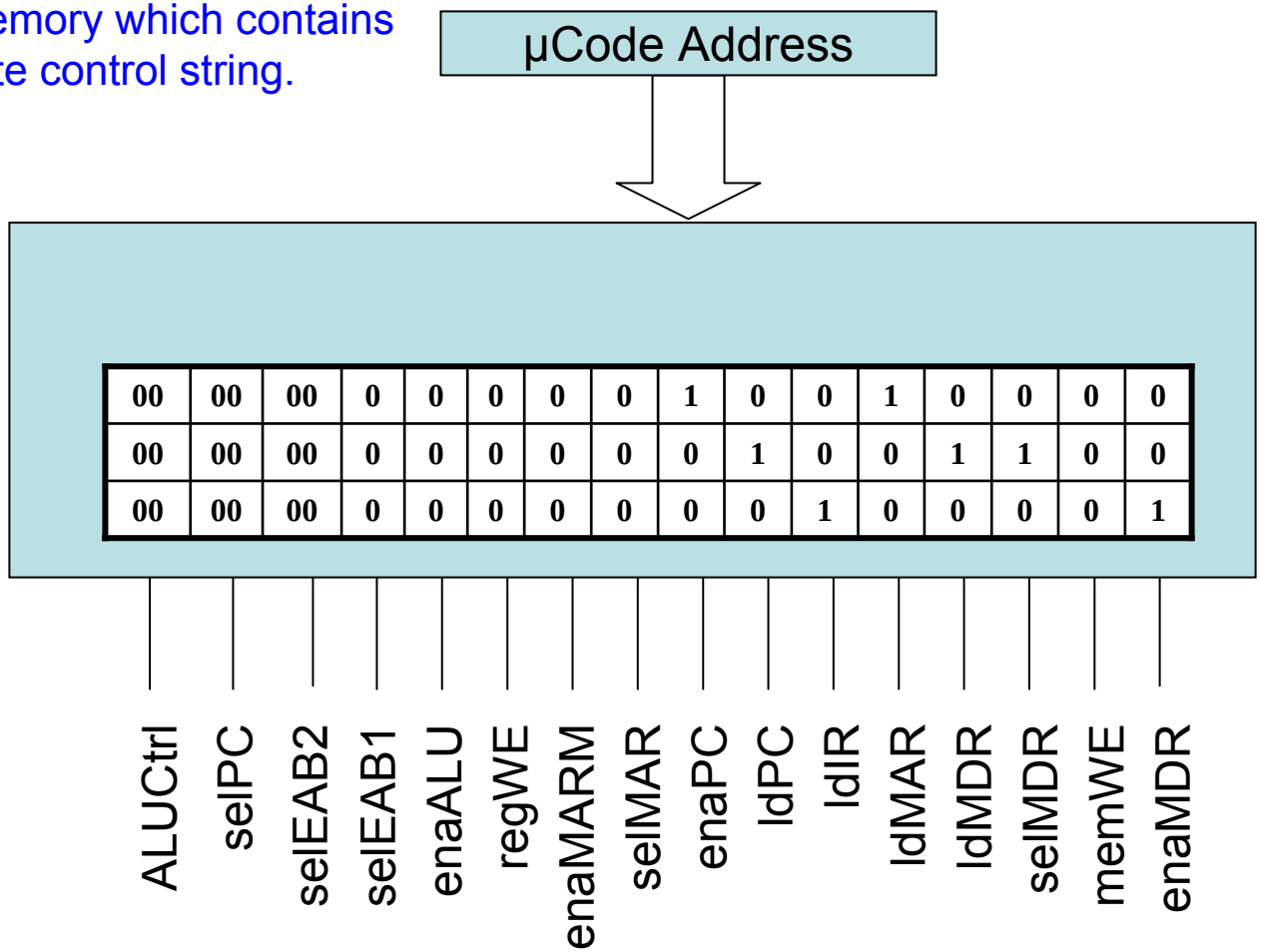
Microprogrammed Control

How do you determine which control string to use?

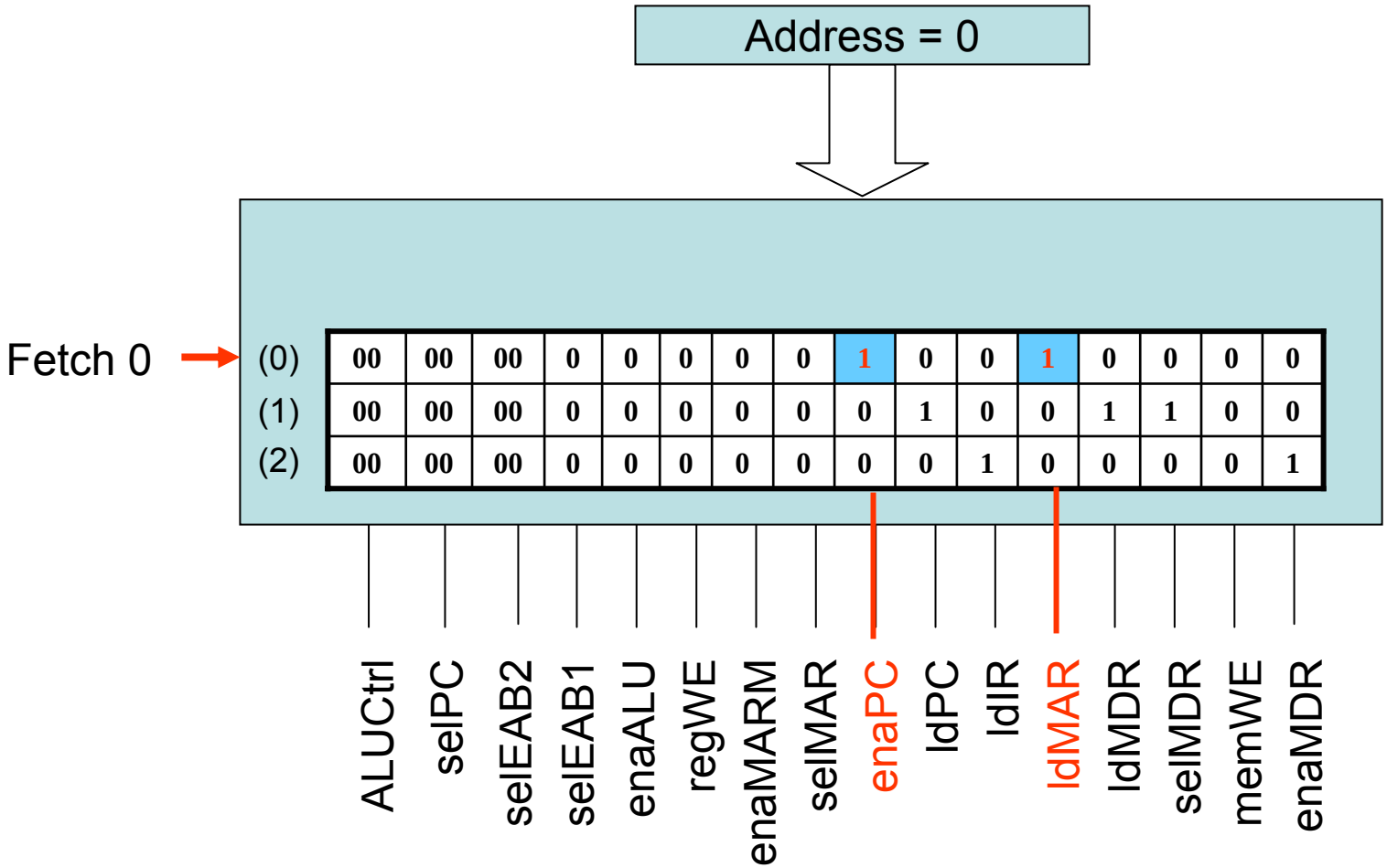


Microprogrammed Control

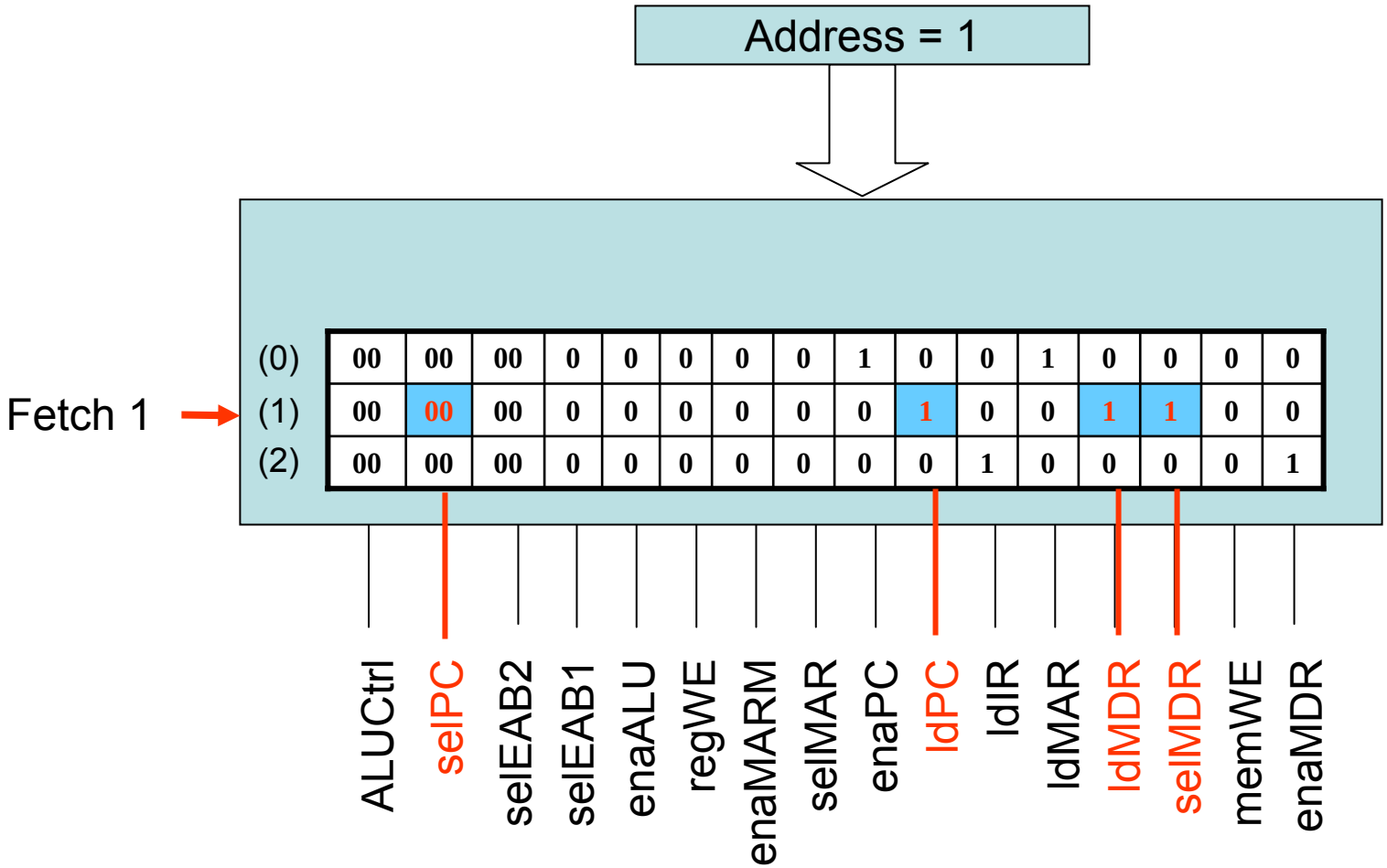
Use a register to select the address in memory which contains the appropriate control string.



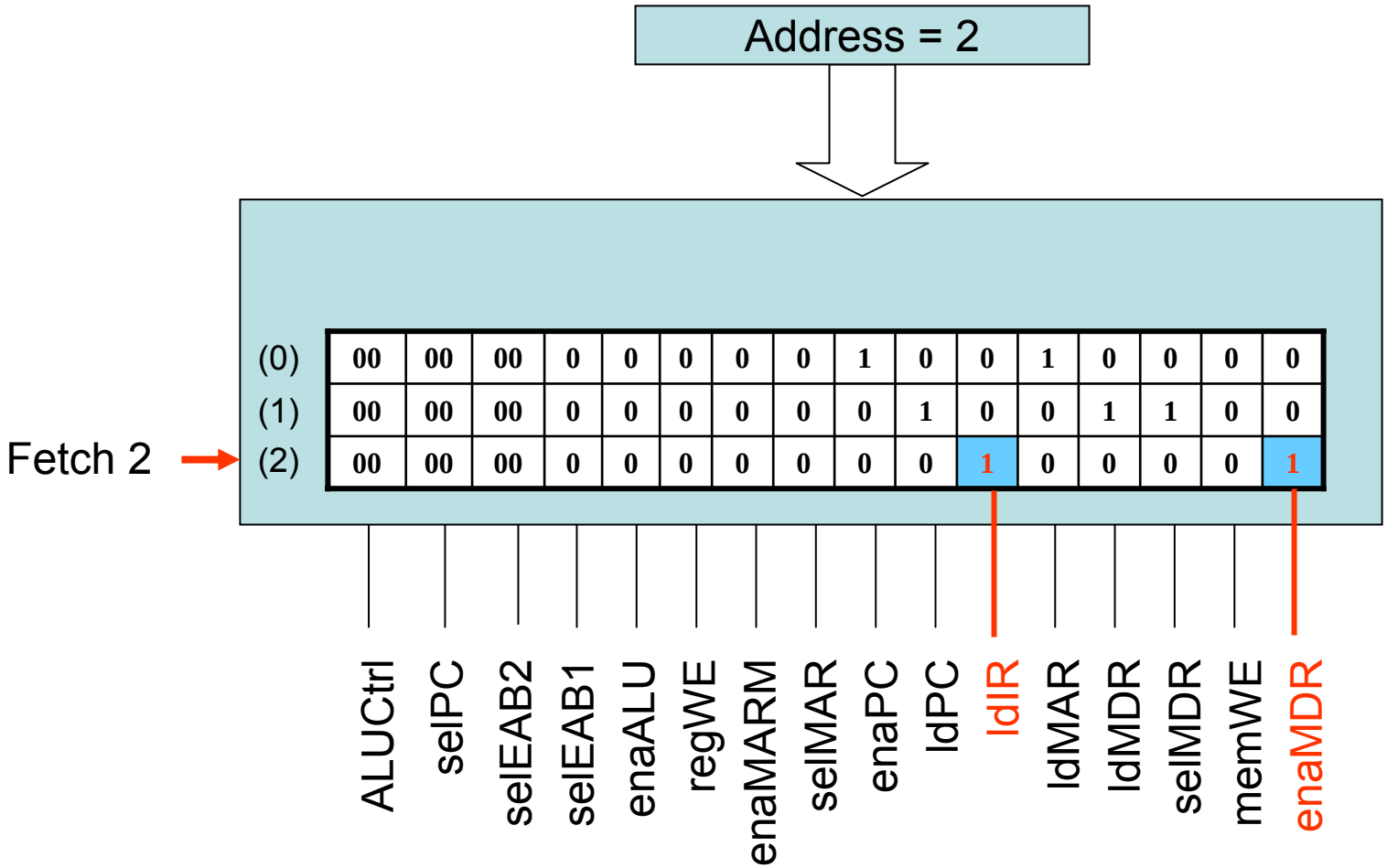
Microprogrammed Control



Microprogrammed Control

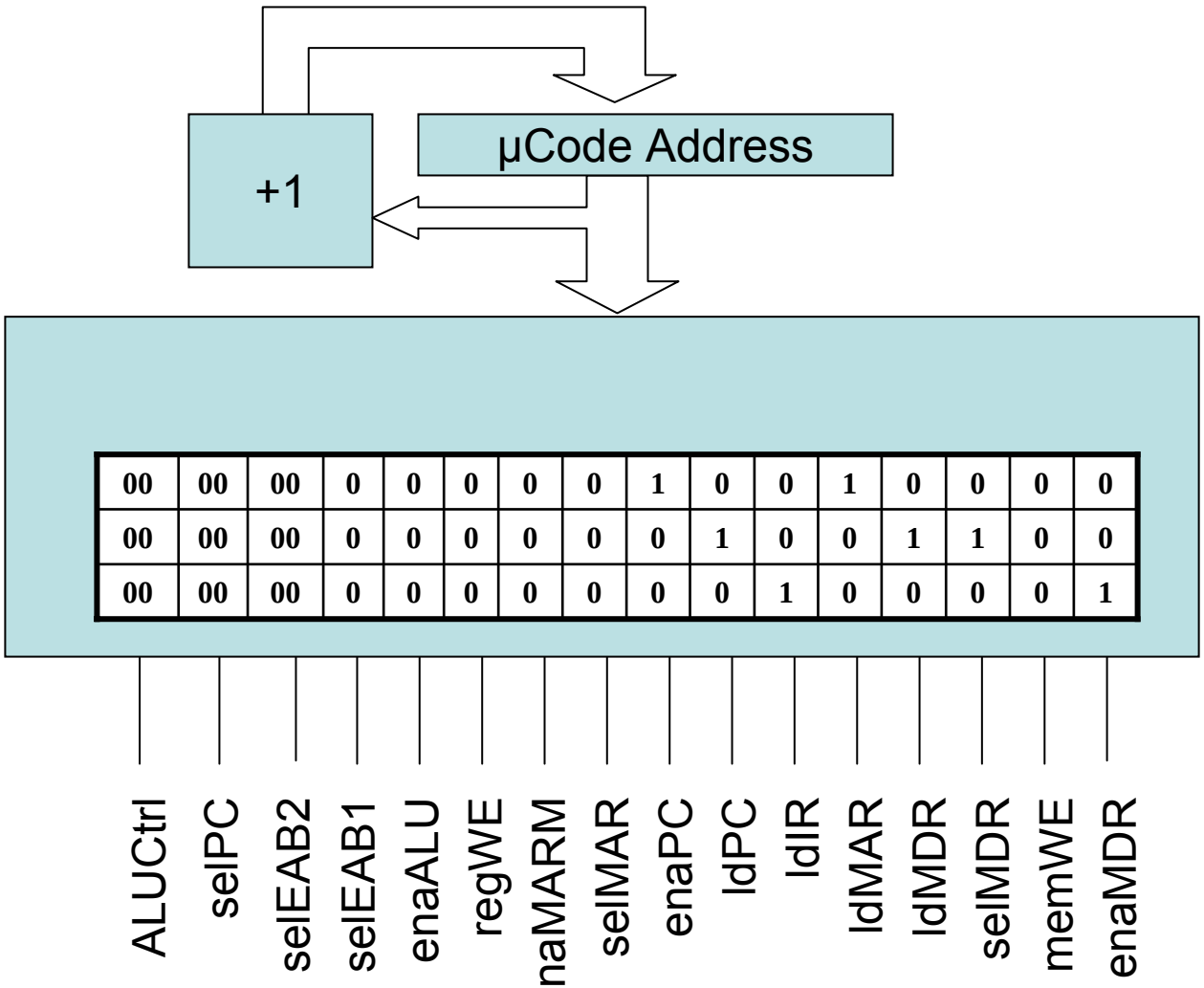


Microprogrammed Control

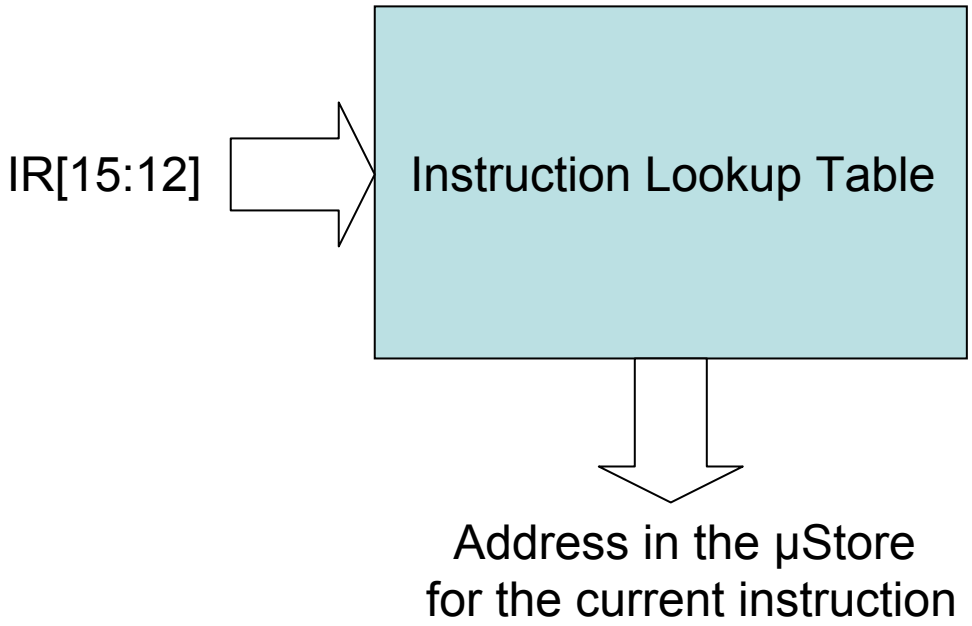


Microprogrammed Control

Use incrementer to get sequential control strings.



Decode

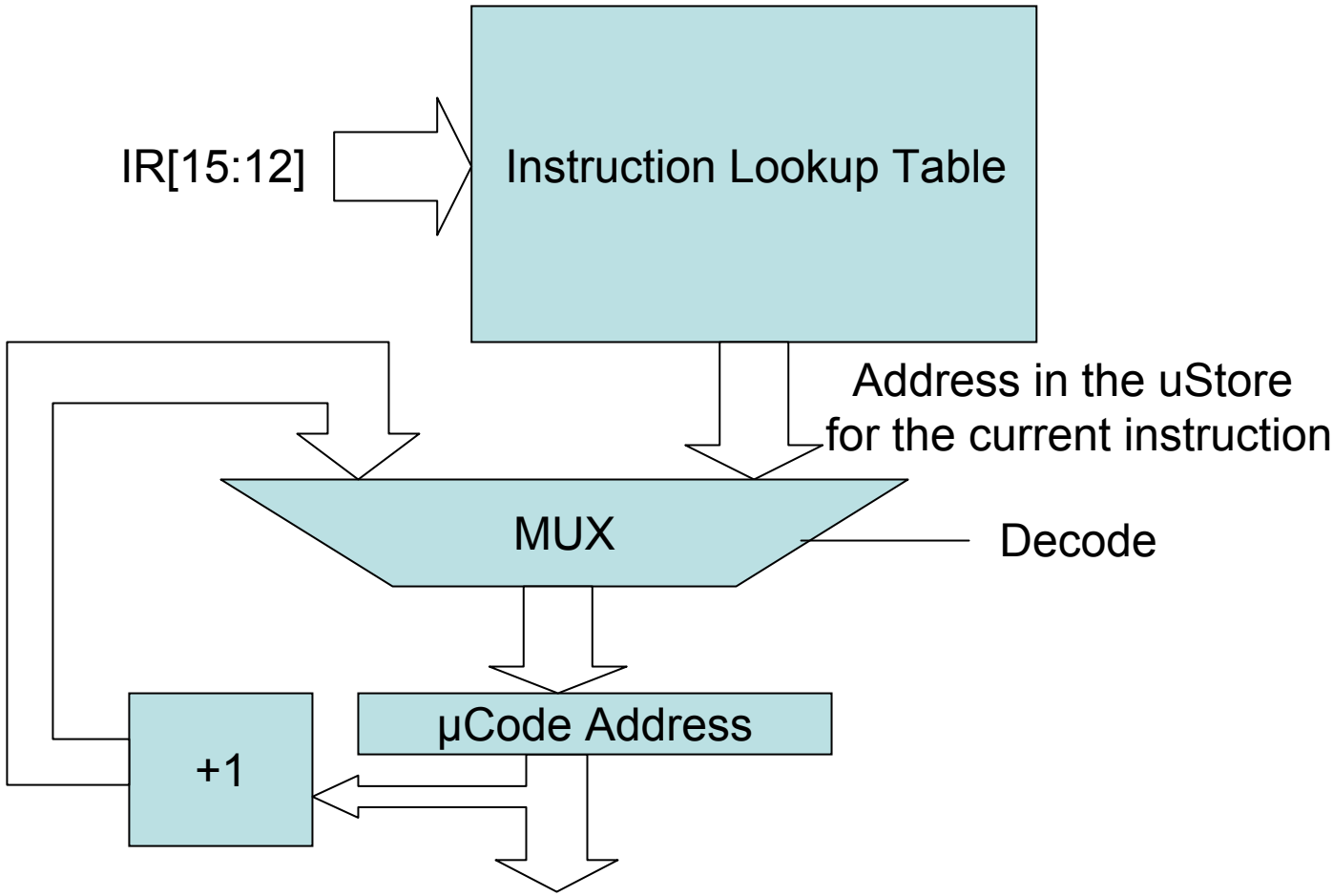


Use a lookup table to get to the appropriate control strings after an instruction has been loaded into the Instruction Register.

This occurs in the Decode stage of the instruction.

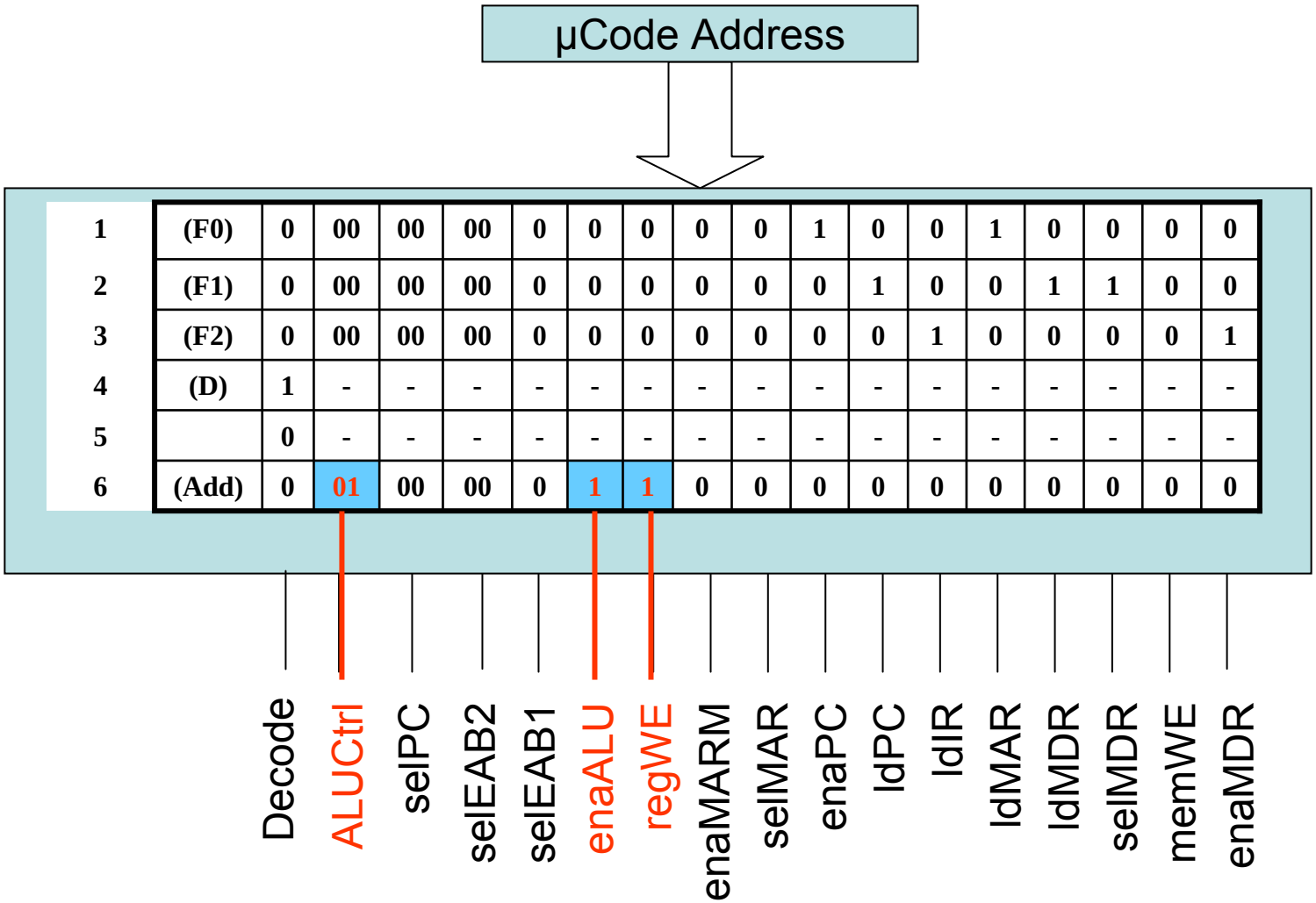
Decode

Use a MUX to select between the sources for the μCode Address.



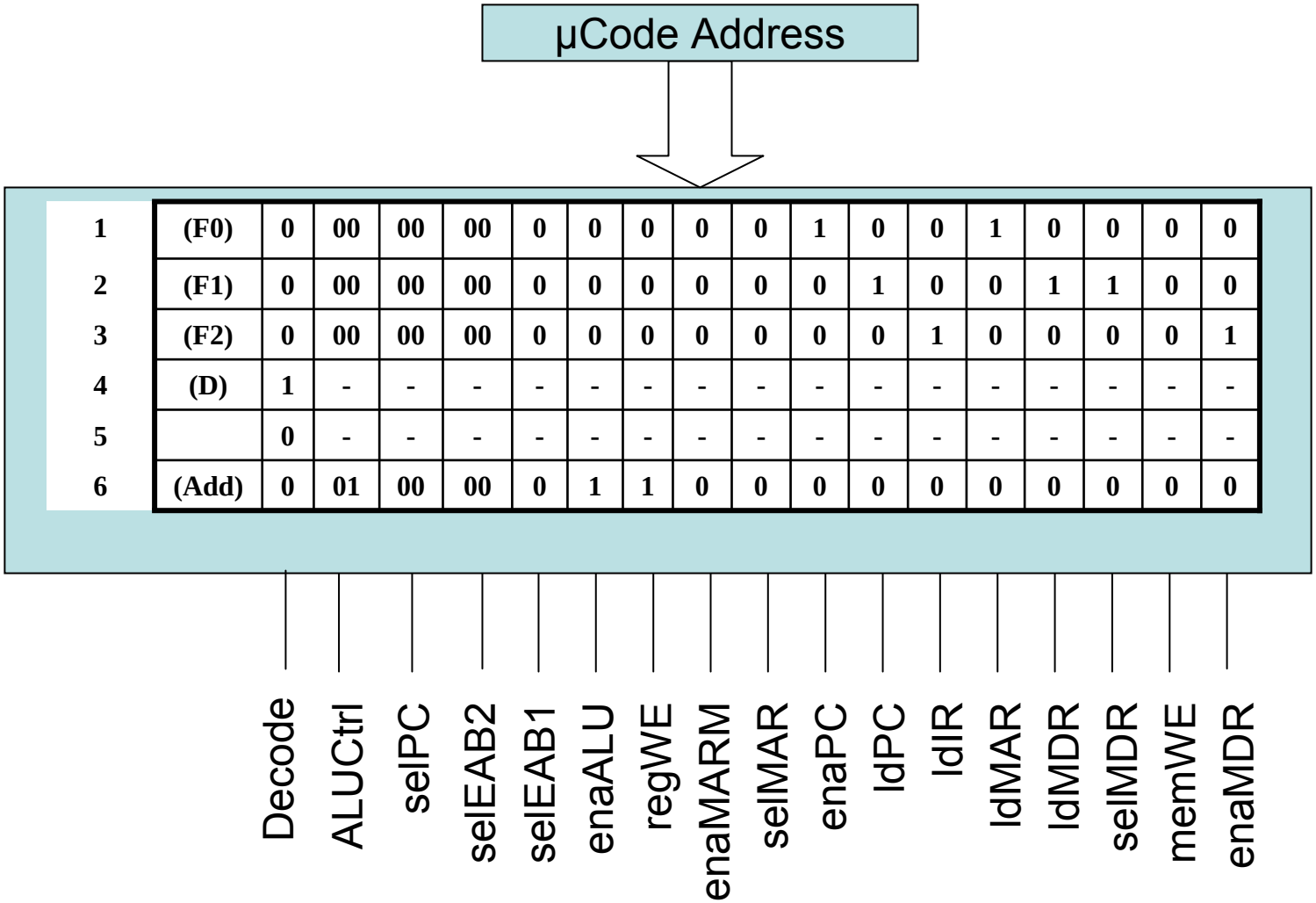
Microprogrammed Control

Instruction Execution



Microprogrammed Control

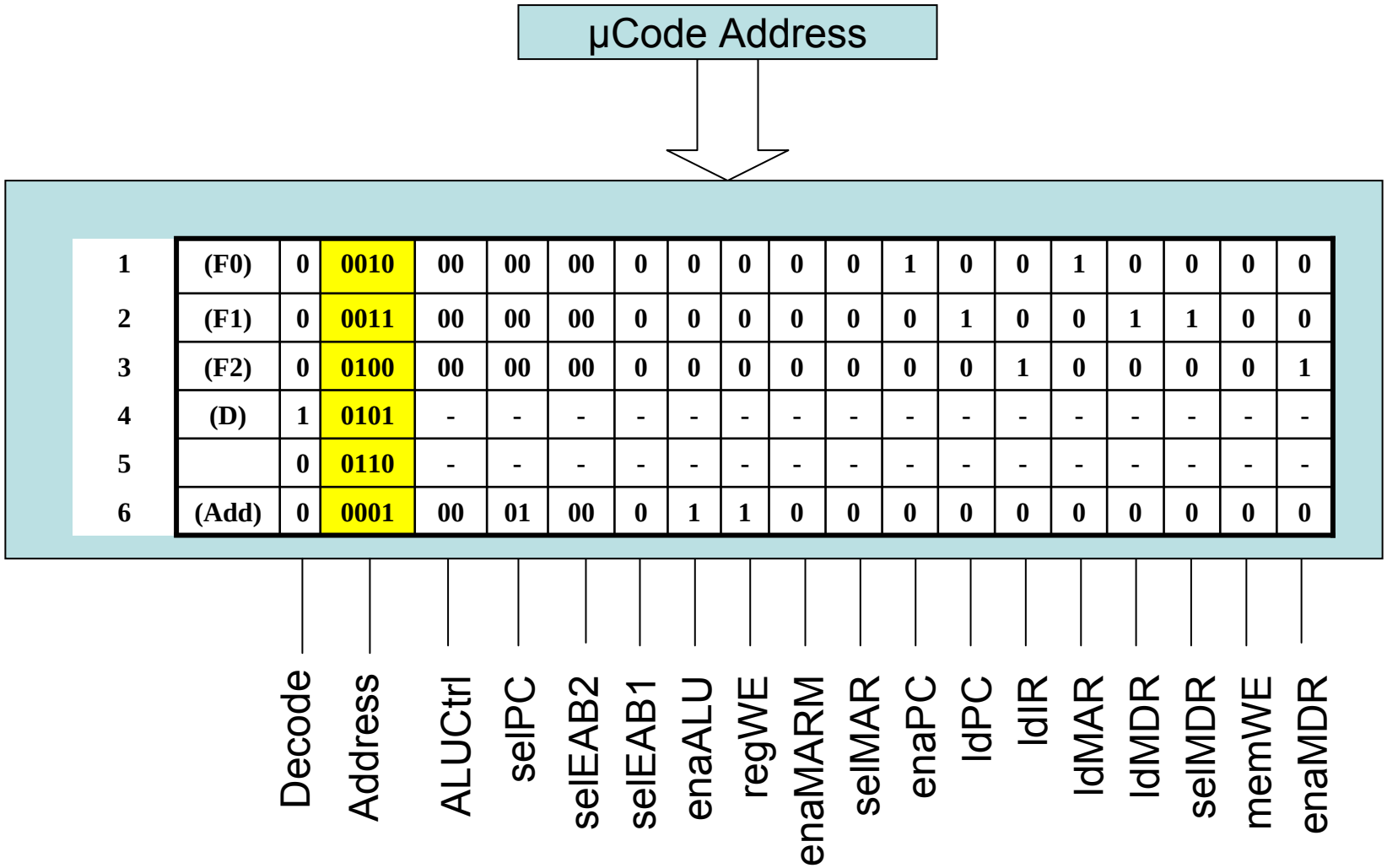
How to return to the fetch cycle?



1	(F0)	0	00	00	00	0	0	0	0	0	1	0	0	1	0	0	0	0
2	(F1)	0	00	00	00	0	0	0	0	0	0	1	0	0	1	1	0	0
3	(F2)	0	00	00	00	0	0	0	0	0	0	0	1	0	0	0	0	1
4	(D)	1	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5		0	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	(Add)	0	01	00	00	0	1	1	0	0	0	0	0	0	0	0	0	0

Microprogrammed Control

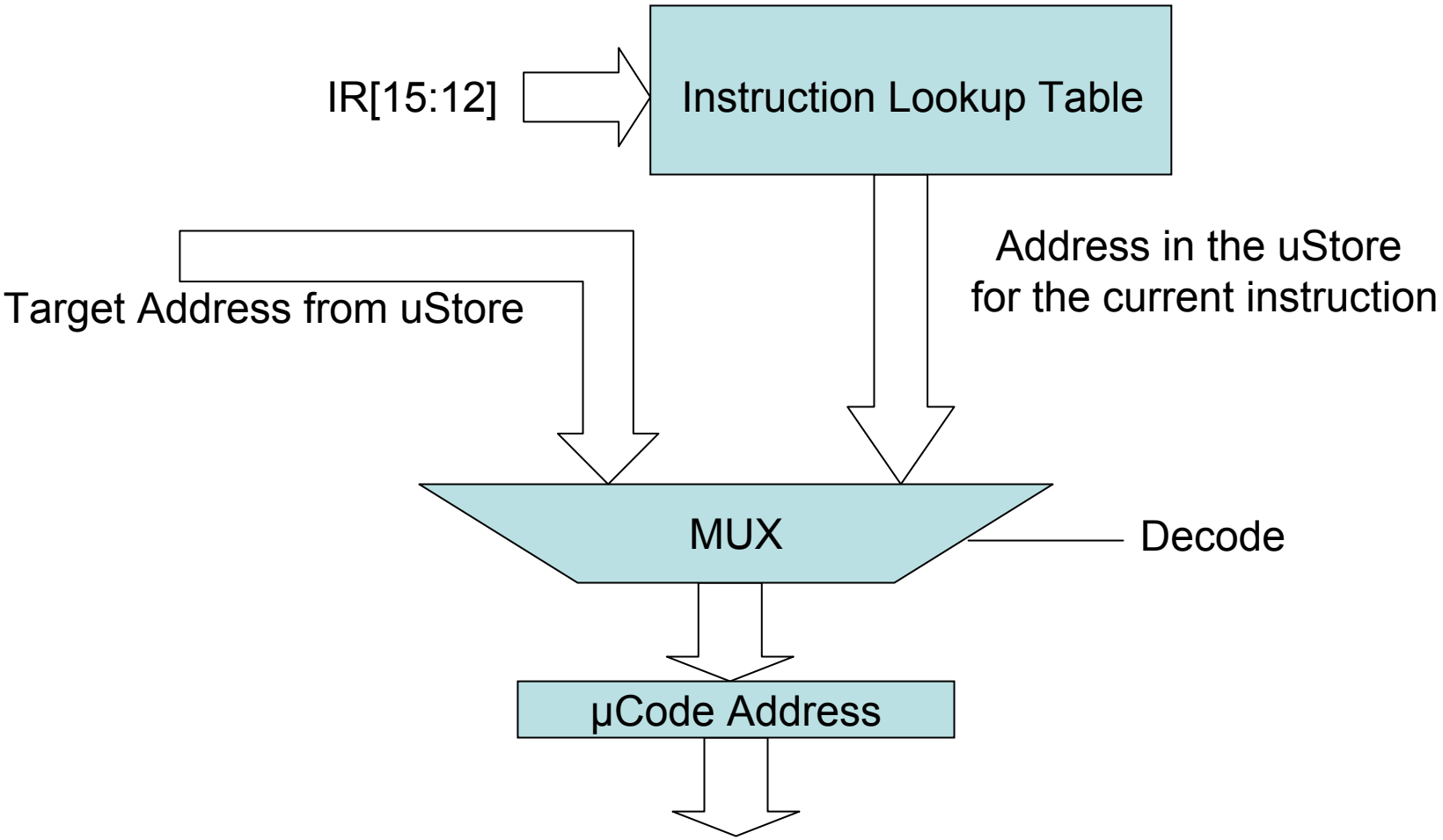
Use a special field to designate the next address which should be executed!



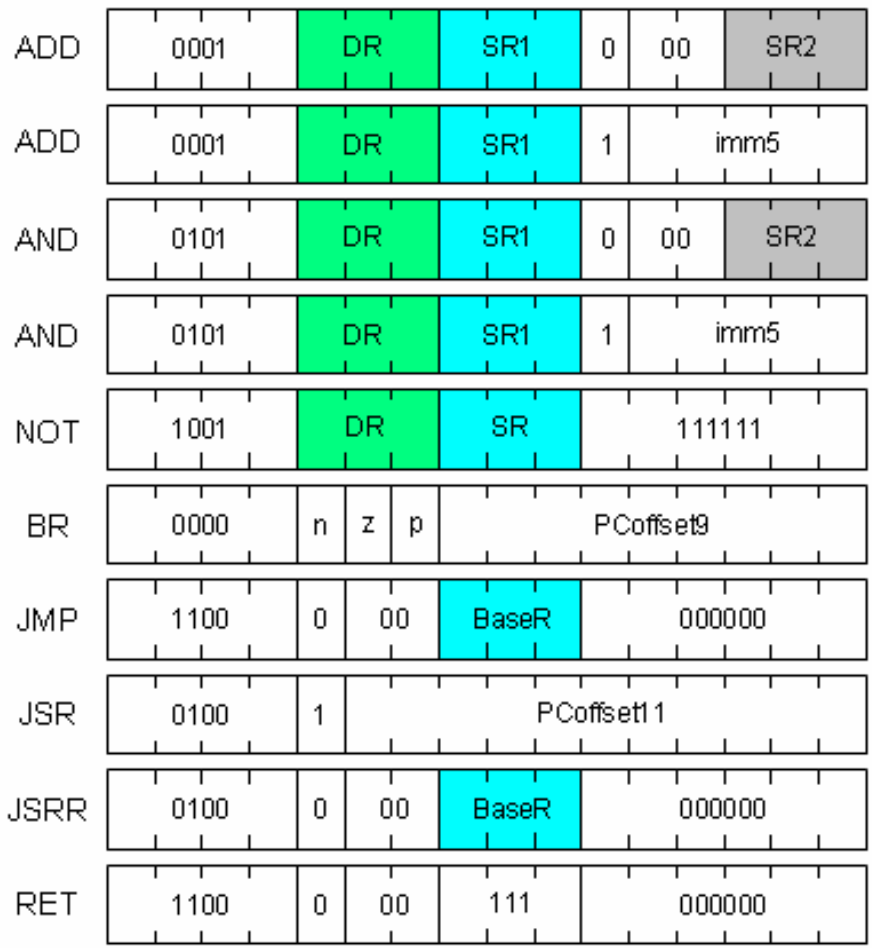
1	(F0)	0	0010	00	00	00	0	0	0	0	0	1	0	0	1	0	0	0	0
2	(F1)	0	0011	00	00	00	0	0	0	0	0	0	1	0	0	1	1	0	0
3	(F2)	0	0100	00	00	00	0	0	0	0	0	0	0	1	0	0	0	0	1
4	(D)	1	0101	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
5		0	0110	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-
6	(Add)	0	0001	00	01	00	0	1	1	0	0	0	0	0	0	0	0	0	0

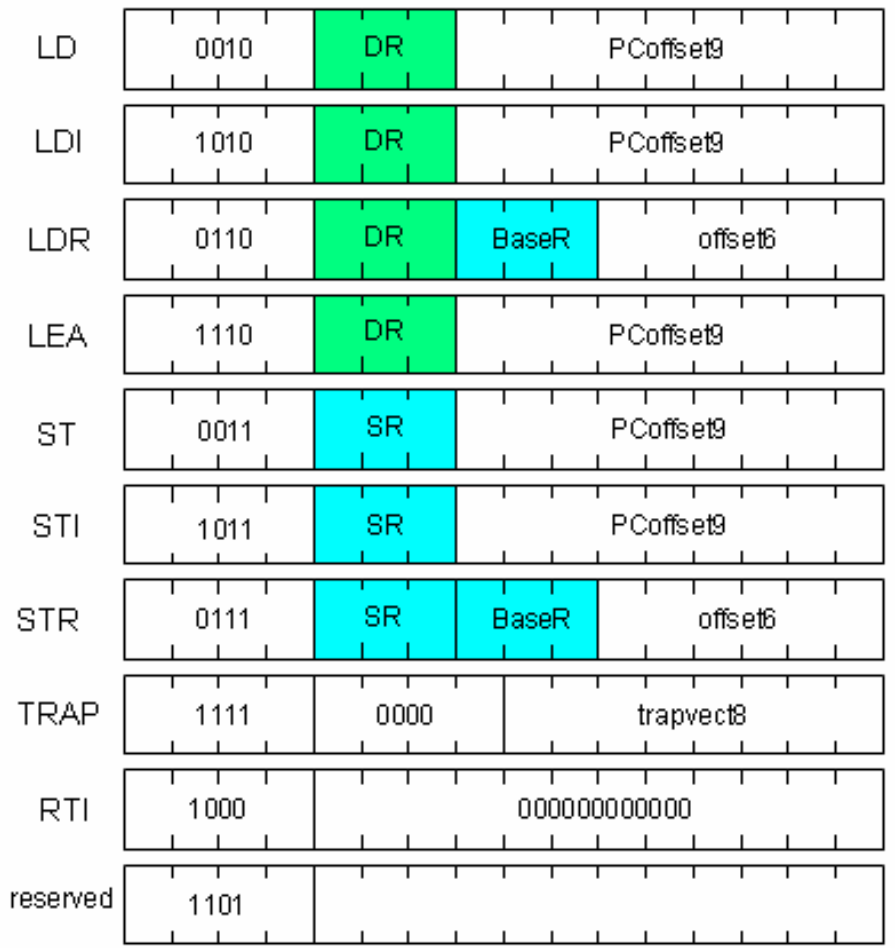
Jump to a New µCode Address

Because of the Target Address field, the incrementer is no longer needed.



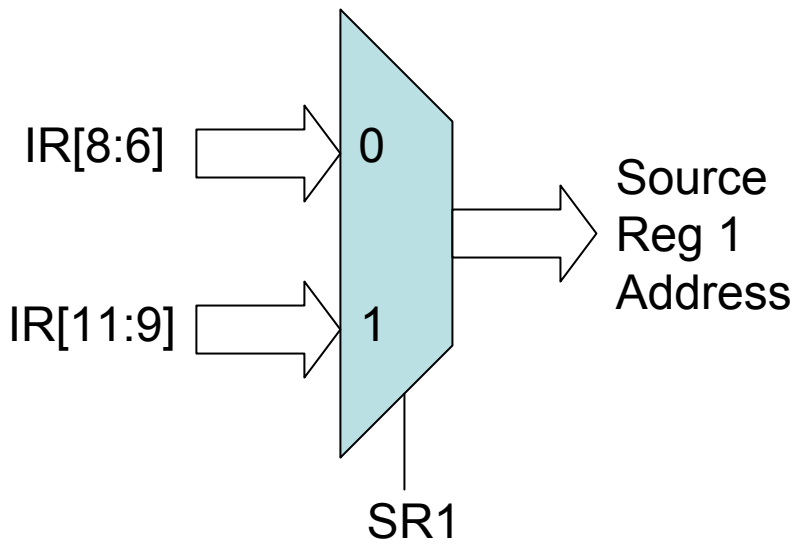
Selecting Operand and Destination Register Addresses

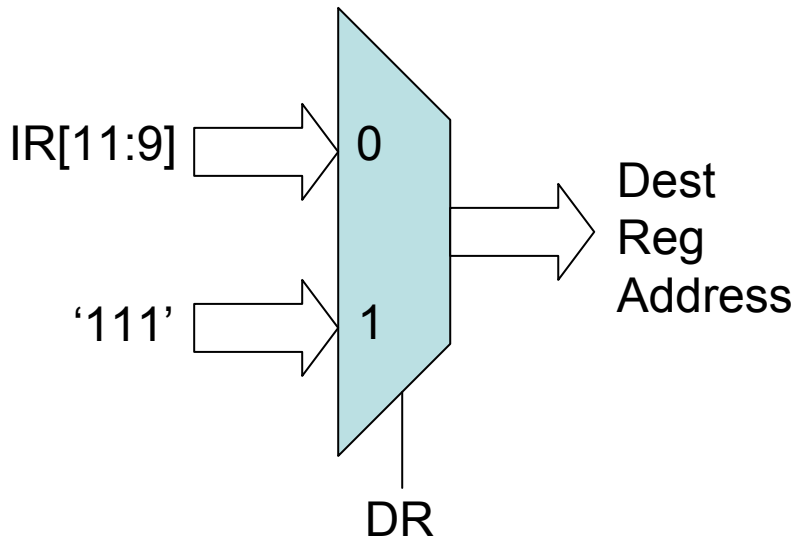




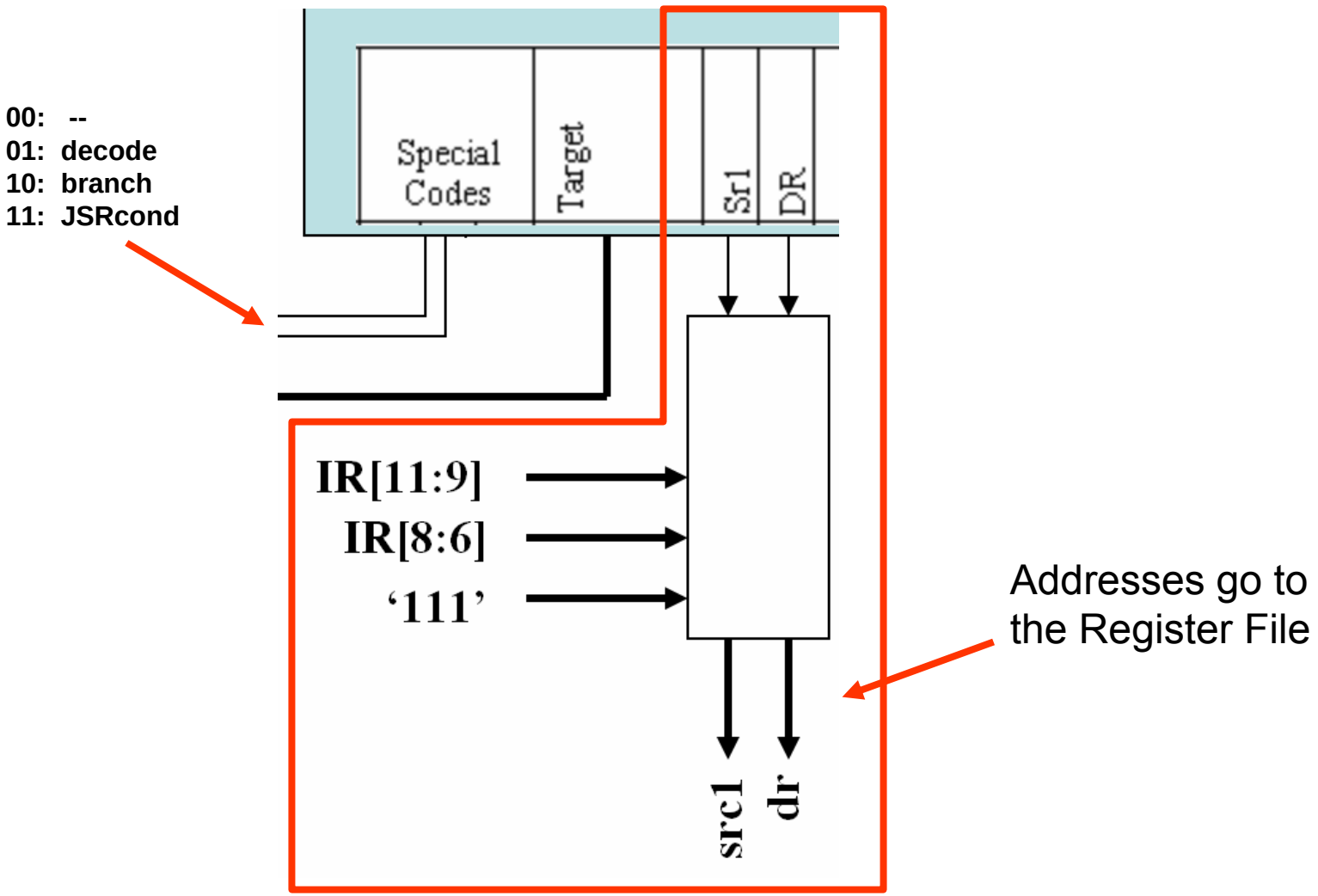
Selecting Operand and Destination Register Addresses

Two more control signals are required in the μStore: SR1 and DR. These signals select the source for the register addresses.





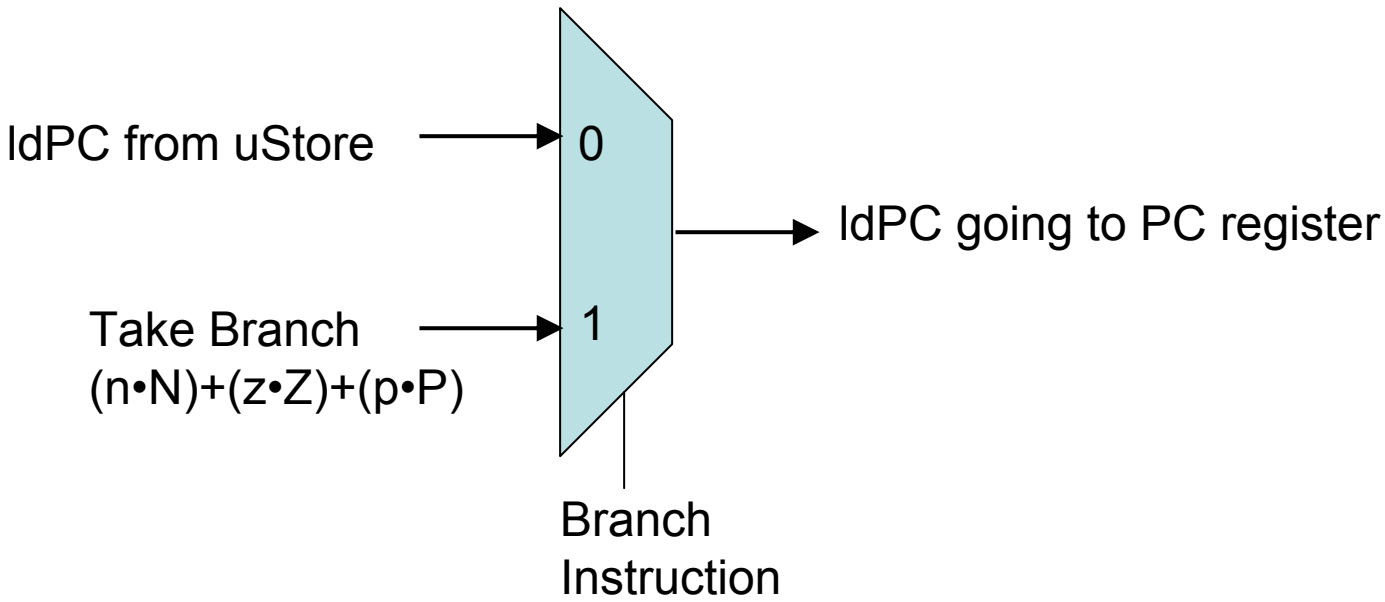
Selecting Operand and Destination Register Addresses



Selecting Load Program Counter Sources

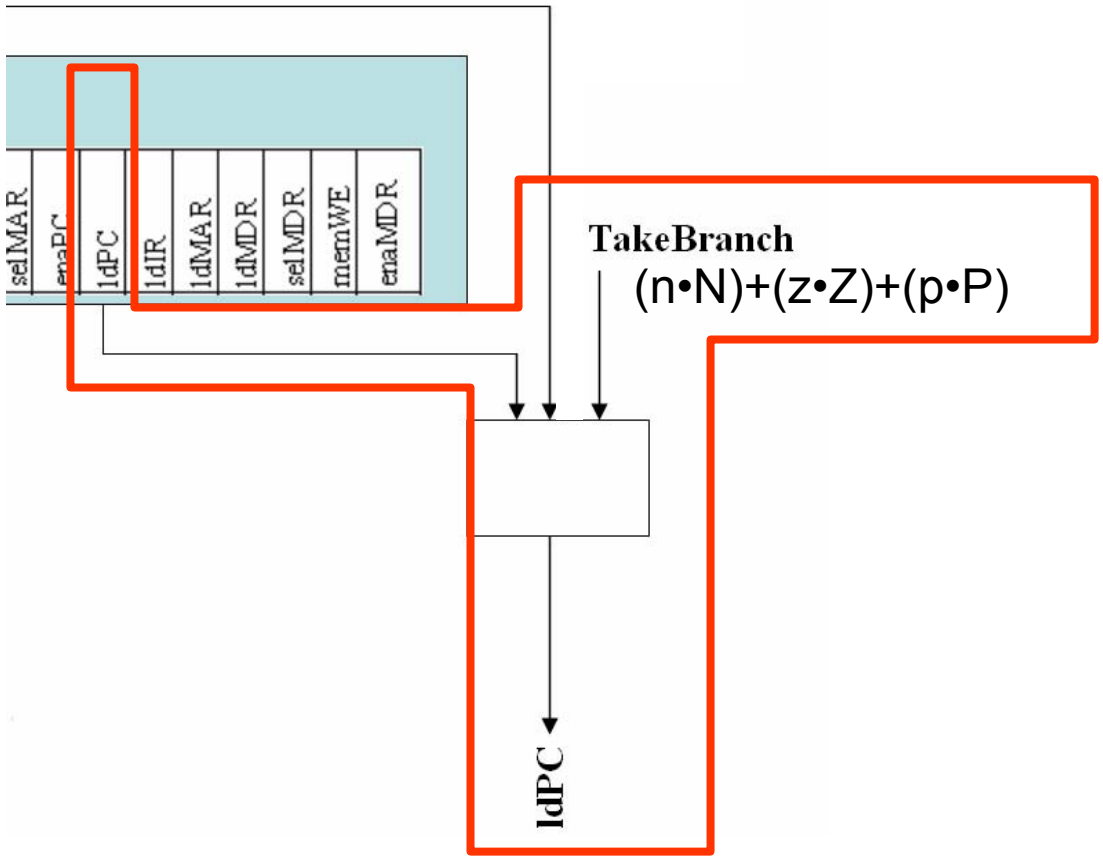
For a Branch instruction, loading of the PC depends upon the N, Z and P register flags and the n, z and p bits in the instruction.

Another control signal is needed to select the source for the load PC control signal.



Selecting Load Program Counter Sources

'Special Code' bit for branch instruction



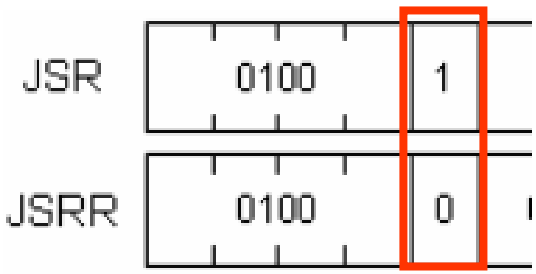
Selecting Jump Subroutine MUX control sources

The two types of JSR instructions use different control signals but share the same opcode.

JSR	**clock1**	R7 ← PC	enaPC ← 1 DR ← 7 regWE ← 1
	clock2	PC ← EAddr	selEAB1← **0** selEAB2 ← **11** selPC ← 01 ldPC← 1
JSRR	**clock1**	R7 ← PC	enaPC← 1 DR← 7 regWE← 1
	clock2	PC ← EAddr	selEAB1← **1** selEAB2 ← **00** selPC ← 01 ldPC← 1

Selecting Jump Subroutine MUX control sources

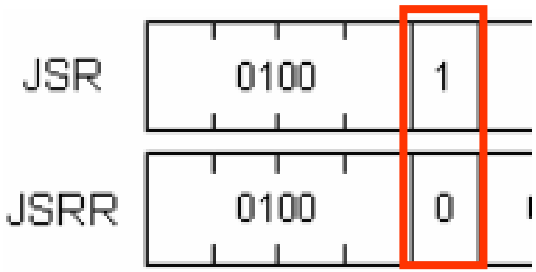
Use IR[11] to distinguish between the two forms of JSR.



JSR	**clock1**	R7 ← PC	enaPC ← 1 DR ← 7 regWE ← 1
	clock2	PC ← EAddr	selEAB1← **0** selEAB2 ← **11** selPC ← 01 ldPC← 1
JSRR	**clock1**	R7 ← PC	enaPC← 1 DR← 7 regWE← 1
	clock2	PC ← EAddr	selEAB1← **1** selEAB2 ← **00** selPC ← 01 ldPC← 1

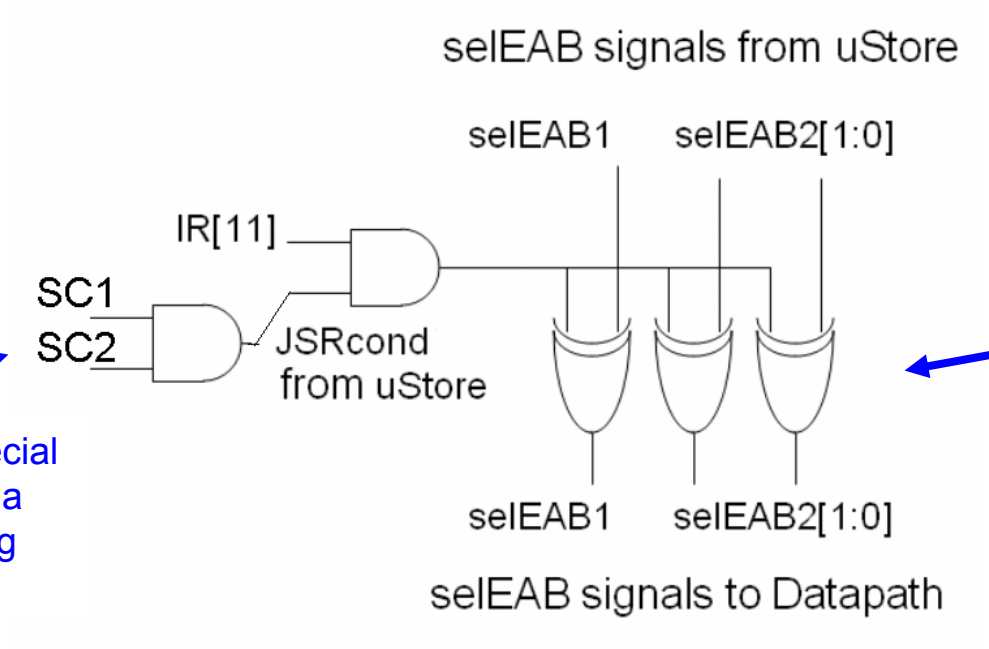
Selecting Jump Subroutine MUX control sources

Luckily, the control signals which differ are the complements of each other.



JSR **clock2** PC ← EAddr selEAB1← **0** selEAB2 ← **11**

JSRR **clock2** PC ← EAddr selEAB1← **1** selEAB2 ← **00**

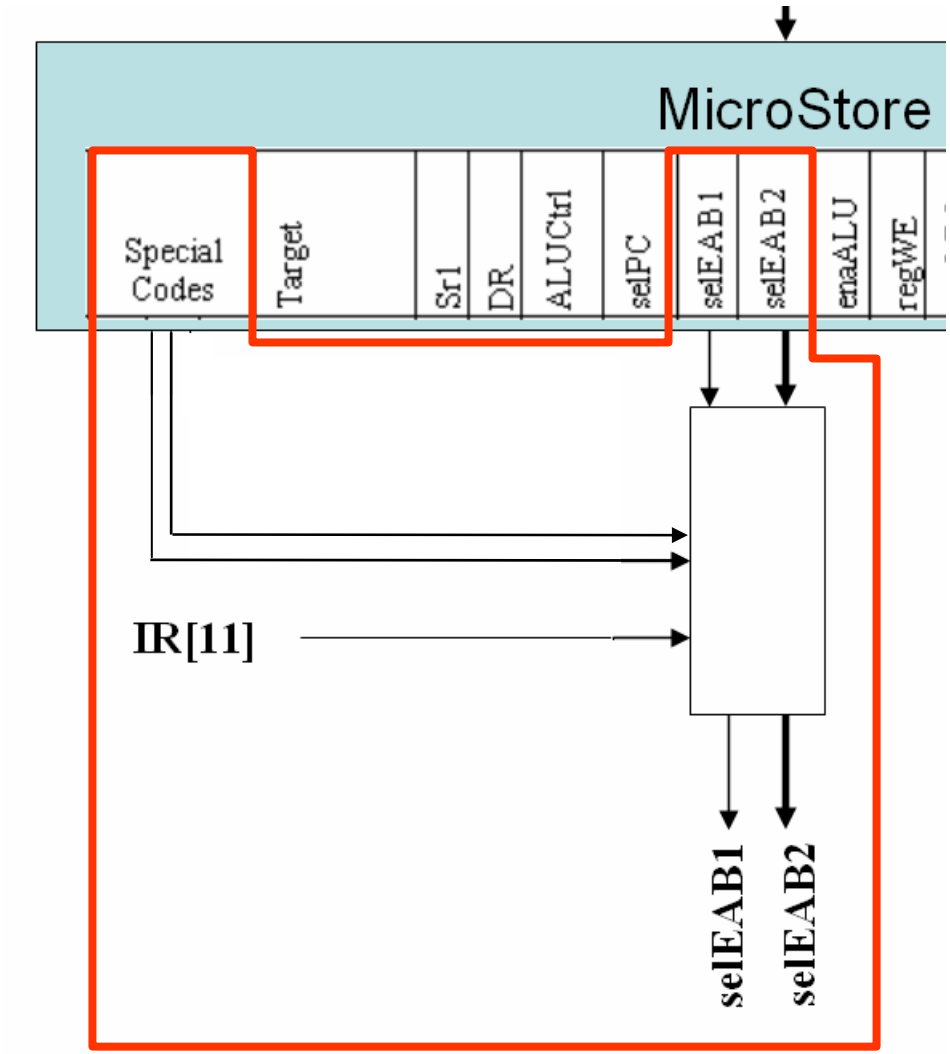


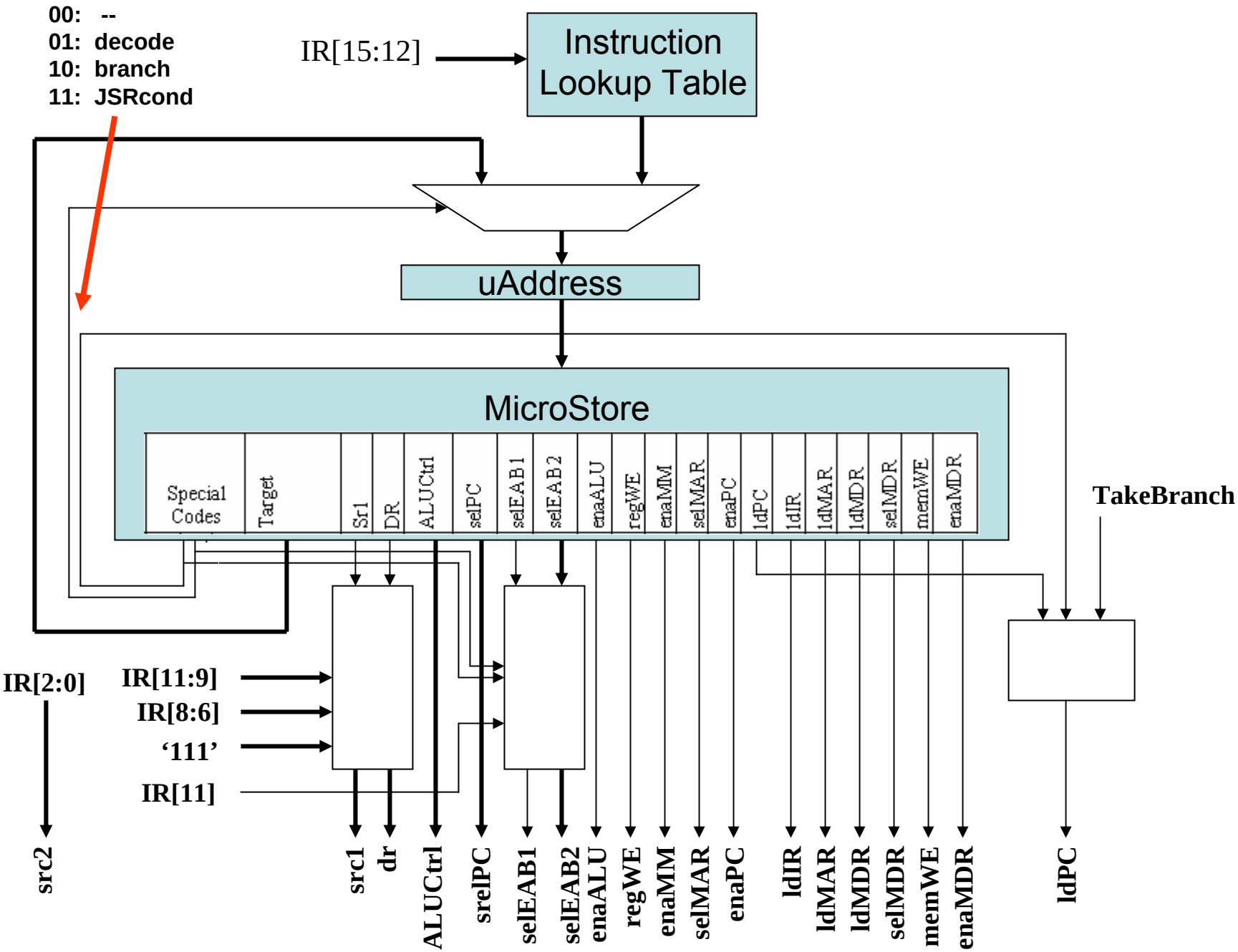
00: --
01: decode
10: branch
11: JSRcond

Again, we require 'Special Code' to specify when a JSR instruction is being executed.

Use XOR gates to perform the selective complement function.

Selecting Jump Subroutine MUX control sources




00: --
01: decode
10: branch
11: JSRcond
IR[15:12]
Instruction Lookup Table
uAddress
MicroStore
Special Codes
Target
Sr1
DR
ALUCtrl
selPC
selEAB1
selEAB2
enaALU
regWE
enaMM
selMAR
enaPC
ldPC
ldIR
ldMAR
ldMDR
selMDR
memWE
enaMDR
TakeBranch
IR[2:0]
IR[11:9]
IR[8:6]
'111'
IR[11]
src2
src1
dr
ALUCtrl
srelPC
selEAB1
selEAB2
enaALU
regWE
enaMM
selMAR
enaPC
ldIR
ldMAR
ldMDR
selMDR
memWE
enaMDR
ldPC

MicroStore

		Special Codes		Target Address	Sr1	DR	ALUCtrl	selPC	selEAB1	selEAB2	enaALU	regWE	enaMM	selMAR	enaPC	ldPC	ldIR	ldMAR	ldMDR	selMDR	memWE	enaMDR
0	Reset	0	0	00001	0	0	00	00	0	00	0	0	0	0	0	0	0	0	0	0	0	0
1	Fetch	0	0	00002	0	0	00	00	0	00	0	0	0	0	1	0	0	1	0	0	0	0
2		0	0	00003	0	0	00	00	0	00	0	0	0	0	0	0	0	0	1	1	0	0
3		0	0	00004	0	0	00	00	0	00	0	0	0	0	0	0	1	0	0	0	0	1
4	Decode	1	0	00000	0	0	00	00	0	00	0	0	0	0	0	0	0	0	0	0	0	0
5	ADD	0	0	00001	0	0	01	00	0	00	1	1	0	0	0	0	0	0	0	0	0	0
6	AND	0	0	00001	0	0	10	00	0	00	1	1	0	0	0	0	0	0	0	0	0	0
7	BR	0	1	00001	0	0	00	01	0	10	0	0	0	0	0	0	0	0	0	0	0	0
8	JMP	0	0	00001	0	0	00	01	1	00	0	0	0	0	0	1	0	0	0	0	0	0
9	JSR/R	0	0	01010	0	1	00	00	0	00	0	1	0	0	1	0	0	0	0	0	0	0
10		1	1	00001	0	0	00	01	1	00	0	0	0	0	0	1	0	0	0	0	0	0
11	LD	0	0	01100	0	0	00	00	0	10	0	0	1	0	0	0	0	1	0	0	0	0
12		0	0	01101	0	0	00	00	0	00	0	0	0	0	0	0	0	0	1	1	0	0
13		0	0	00001	0	0	00	00	0	00	0	1	0	0	0	0	0	0	0	0	0	1
14	LDI	0	0	01111	0	0	00	00	0	10	0	0	1	0	0	0	0	1	0	0	0	0
15		0	0	10000	0	0	00	00	0	00	0	0	0	0	0	0	0	0	1	1	0	0
16		0	0	01100	0	0	00	00	0	00	0	0	0	0	0	0	0	1	0	0	0	1
17	LDR	0	0	01100	0	0	00	00	1	01	0	0	1	0	0	0	0	1	0	0	0	0
18	LEA	0	0	00001	0	0	00	00	0	10	0	1	1	0	0	0	0	0	0	0	0	0
19	NOT	0	0	00001	0	0	11	00	0	00	1	1	0	0	0	0	0	0	0	0	0	0
20	ST	0	0	10101	0	0	00	00	0	10	0	0	1	0	0	0	0	1	0	0	0	0
21		0	0	10110	1	0	00	00	0	00	1	0	0	0	0	0	0	0	1	0	0	0
22		0	0	00001	0	0	00	00	0	00	0	0	0	0	0	0	0	0	0	0	1	0
23	STI	0	0	11000	0	0	00	00	0	10	0	0	1	0	0	0	0	1	0	0	0	0
24		0	0	11001	0	0	00	00	0	00	0	0	0	0	0	0	0	0	1	1	0	0
25		0	0	10101	0	0	00	00	0	00	0	0	0	0	0	0	0	1	0	0	0	1
26	STR	0	0	10101	0	0	00	00	1	01	0	0	1	0	0	0	0	1	0	0	0	0

Instruction Lookup Table

Op-code	Instruction	Next Address (decimal)	Next Address (binary)
0000	BR	7	00111
0001	ADD	5	00101
0010	LD	11	01011
0011	ST	20	10100
0100	JSR	9	01001
0101	AND	6	00110
0110	LDR	17	10001
0111	STR	26	11010
1000	RTI	----	---
1001	NOT	19	10011
1010	LDI	14	01110
1011	STI	23	10111
1100	JMP	8	01000
1101	----	----	----
1110	LEA	18	10010
1111	TRAP	----	----