Debdeep Mukhopadhyay,
CSE, IIT Kharagpur

The Micro Mips ISA

Class	Instruction	Usage	Meaning	op	fn
Copy	Load upper immediate	lui rt, imm	$rt \leftarrow (imm, 0x0000)$	15	
Arithmetic	Add	add rd,rs,rt	$rd \leftarrow (rs) + (rt)$	0	32
	Subtract	sub rd,rs,rt	$rd \leftarrow (rs) - (rt)$	0	34
	Set less than	slt rd,rs,rt	$rd \leftarrow$ if $(rs) < (rd)$ then 1 else 0	0	42
	Add Immediate	addi rt,rs,imm	$rt \leftarrow (rs) + imm$	8	
	Set Less than immediate	slti rt,rs,imm	$rt \leftarrow if(rs) < imm$then 1 else 0	10	
Logic	AND	and rd,rs,rt	$rd \leftarrow (rs) \wedge (rt)$	0	36
	OR	or rd,rs,rt	$rd \leftarrow (rs) \vee (rt)$	0	37
	XOR	xor rd,rs,rt	$rd \leftarrow (rs) \oplus (rt)$	0	38
	NOR	nor rd,rs,rt	$rd \leftarrow ((rs) \vee (rt))'$	0	39
	AND immediate	andi rt,rs,imm	$rt \leftarrow (rs) \wedge imm$	12	
	OR immediate	ori rt,rs,imm	$rt \leftarrow (rs) \vee imm$	13	
	XOR immediate	xori rt,rs,imm	$rt \leftarrow (rs) \oplus imm$	14	
Memory Word	Load Word	lw rt,imm(rs)	$rt \leftarrow mem[(rs) + imm]$	35	
	Store Word	sw rt,imm,(rs)	$mem[(rs) + imm] \leftarrow (rt)$	43	
Control transfer	Jump	j L	goto L	2	
	Jump register	jr rs	goto (rs)	0	8
	Branch on less than 0	bltz rs,L	if$(rs) < 0$ then goto L	1	
	Branch on equal	beq rs,rt,L	if $(rs) = (rt)$ then goto L	4	
	Branch on not equal	bne rs,rt,L	if$(rs) \neq (rt)$ then goto L	5	
	Jump and link	jal L	goto L; 31 ← (PC)+4	3	
	System call	syscall	Associated with an OS system routine	0	12

The Instruction Format

op	rs	rt	rd	sh	fn
6 bits	**5 bits**	**5 bits**	**5 bits**	**5 bits**	**6 bits**
Opcode	**Source 1 or base**	**Source 2 or dest**	**Destination**	**Unused imm**	**Opcode ext**

Operand.offset,16 bits (rd, sh, fn)

jta

jump target address (rs through fn)

Performance of the Single Cycle Architecture

- The above design of control circuit is a stateless and combinational design.
- Each new instruction is read from the PC, and is executed in one single clock.
  - Thus CPI=1
- The clock cycle is determined by the longest instruction.

lw is the longest instruction

- lw execution includes all the possible steps:
  1. Instruction Excess: 2 ns
  2. Register Read: 1 ns
  3. ALU operation: 2 ns
  4. Data Cache Access: 2 ns
  5. Register Write-back: 1 ns

     Total: 8 ns

  Thus a clock frequency of 125 MHz suffices.

  So, for 1 instruction, (1/125) x 10^{-6} sec

  Thus, 125 Million Instructions are executed per second (125 MIPS)

Obtaining better performance

- Note that the average instruction time is less, depends on the type of instruction, and their percentages in an application.
- Rtype 44% 6 ns No data cache

  Load 24% 8 ns

  Store 12% 7ns No register write-back

  Branch 18% 5ns Fetch+Register Read+Next-addr formation

  Jump 2% 3ns Fetch + Instruction Decode

  Weighted average = 6.36 ns

So, with a variable cycle time implementation, the performance is 157 MIPS

However, this is not possible. But we see that a single cycle implementation has a poor performance.

Summary

- Clock cycle is determined by the slowest instruction.
- If the MIPS ISA includes more complex instructions, the disadvantage is more.
  - For example if we add a MULT/DIV instruction, then all operations need to be slowed down.
  - Thus MIPS does the MIPS/DIV instruction to a separate block (than the ALU block), with separate registers Hi and Lo.
    - sufficient time is kept to write back the results to the register file

Shorter Clock Cycles in Multi-cycle implementation

- The MIPS instructions typically has a set of actions, namely: memory access, register read, ALU operation, register write back.
- Each takes around 2 ns time.
- In a single cycle implementation, the worst-case (longest) time of the instructions is taken as the clock frequency.
- In a multi-cycle implementation, a subset of these actions is performed in one clock: thus the clock cycle can be much shorter.
- Every instructions takes several clock cycles (thus CPI $\neq 1$)

Comparision between the two approaches

- Consider the execution of n instructions, with the following characteristics

Name	Time needed	No of basic operations
Instruction 1	t_1	i_1
...		
Instruction 2	t_n	i_n

Say, the $\max(t_1,\ldots,t_n)=t$, and each basic operation takes t' time units.

Comparision between the two approaches

- Single Cycle: Clock Period : t
  Total time = nt
- Multi Cycle: Clock Period: t'
  Total time = $(i_1+\ldots+i_n)t'$

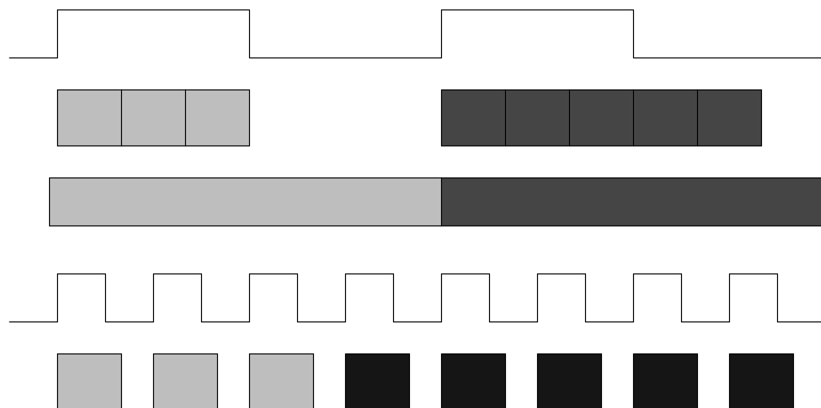
Thus, multi-cycle is better if:

$(i_1+\ldots+i_n)t' < nt$

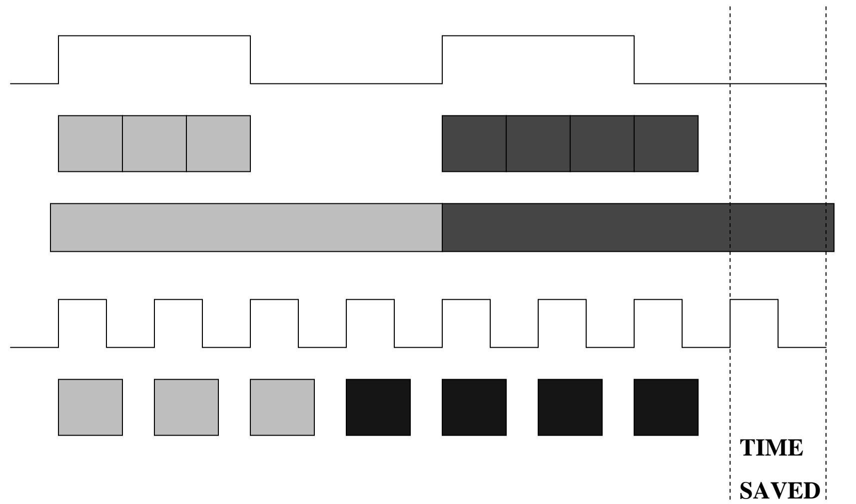
or, $(i_1+\ldots+i_n) < n(t/t')$

or, $I < nr$

I=8, n=2, r=4



I=7, n=2, r=4



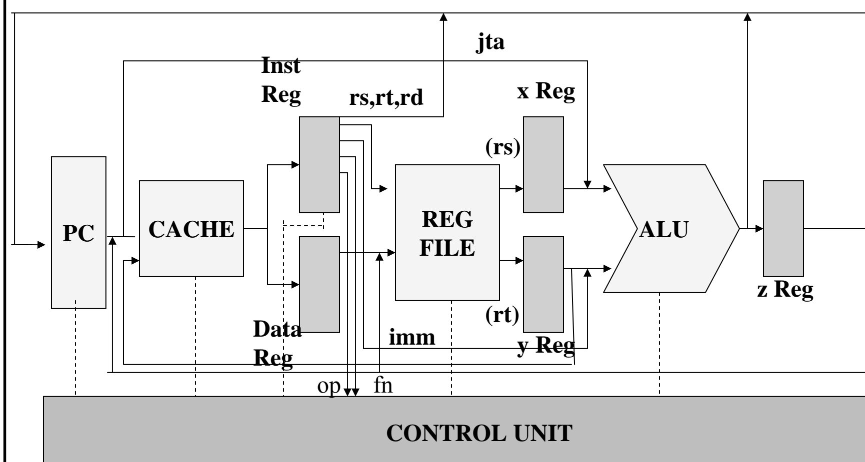
Multi-cycles of the Instructions

- Each instruction starts in the same way (at the same state) and passes through 3-5 clock cycles before being executed:
  1. Instruction Fetch Cycle
  2. Instruction Decode and Register Access
  3. update of PC (Jump/Branch), ALU operations: (-) in case of branch, (+) in case of lw/sw, varies (in case of ALU-type instructions)
  4. Memory Read (lw), Memory Write (sw)
  5. Register Write Back (lw)

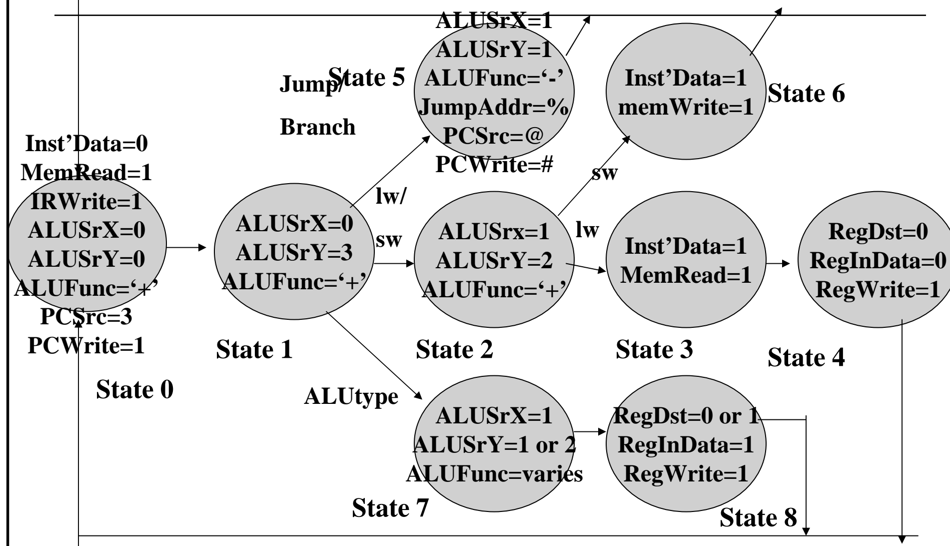
Subtle Points/Differences from the single cycle implementation

- A single memory unit suffices (as read and write from and to memory) are at different clock cycles.
- Requirement of Instruction Register: This register has to hold the instructions to generate appropriate control signals through the multiple cycles until it is executed.

Abstraction of Instruction Execution Unit



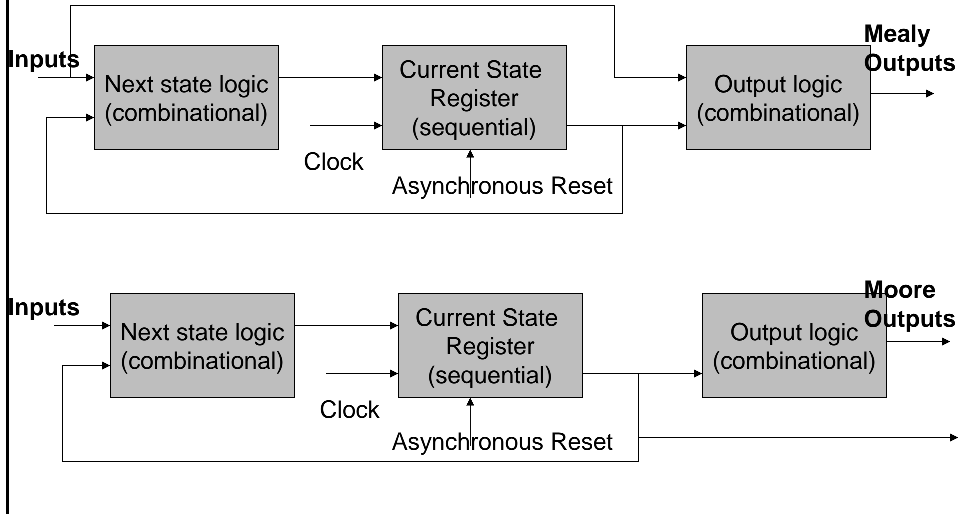
The control state machine



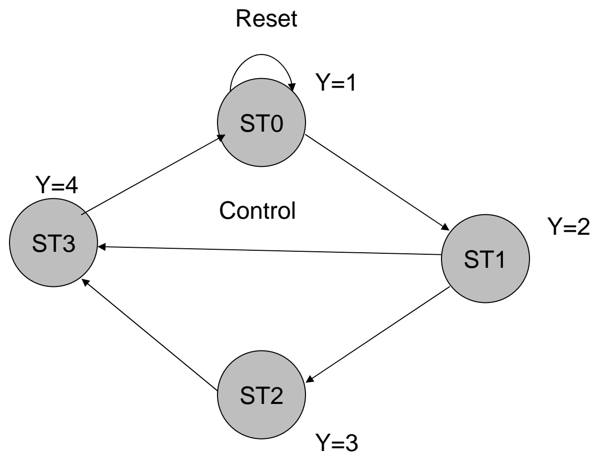
State 5

- %: 0 for j or jal, 1 for syscall, don't care for other instructions
- @: 0 for j, jal, syscall, 1 for jr, 2 for branches
- #: 1 for j, jr, jal, syscall, ALUzero(') for beq(bne),bit 31 of ALUout for bltz
- For jal, RegDst=2, RegInData=1, RegWrite=1

FSM Types



Coding FSMs in Verilog



Issues

- State Encoding
  - sequential
  - gray
  - Johnson
  - one-hot

Encoding Formats

No	Sequential	Gray	Johnson	One-hot
0	000	000	0000	00000001
1	001	001	0001	00000010
2	010	011	0011	00000100
3	011	010	0111	00001000
4	100	110	1111	00010000
5	101	111	1110	00100000
6	110	101	1100	01000000
7	111	100	1000	10000000

Comments on the coding styles

- **Binary:** Good for arithmetic operations. But may have more transitions, leading to more power consumptions. Also prone to error during the state transitions.
- **Gray:** Good as they reduce the transitions, and hence consume less dynamic power. Also, can be handy in detecting state transition errors.

Coding Styles

- **Johnson:** Also there is one bit change, and can be useful in detecting errors during transitions. More bits are required, increases linearly with the number of states. There are unused states, so we require either explicit asynchronous reset or recovery from illegal states (even more hardware!)
- **One-hot:** yet another low power coding style, requires more no of bits. Useful for describing bus protocols.

Improper way

```
always @(posedge Clock or posedge Reset)
begin
   if(Reset) begin
       Y=1;
       STATE=ST0;
    end
```

Improper Way leads to unnecessary latches

```
else
   case(STATE)
      ST0: begin Y=1; STATE=ST1; end
      ST1: begin Y=2;
              if(Control)  STATE=ST2;
              else STATE=ST3;
       ST2: begin Y=3; STATE=ST3; end
       ST3: begin Y=4; STATE=ST0; end
    endcase
  end
```

Output Y is assigned under synchronous always block
so extra latches inferred.

Good FSMs

- Keep separate CS, NS and OL

Next State (NS)

```
always @(input or currentstate)
begin
    NextState=ST0;
    case(currentstate)
      ST0: begin
               NextState=ST1;
             end
      ST1: begin …
      …
      ST3:
               NextState=ST0;
  endcase
end
```

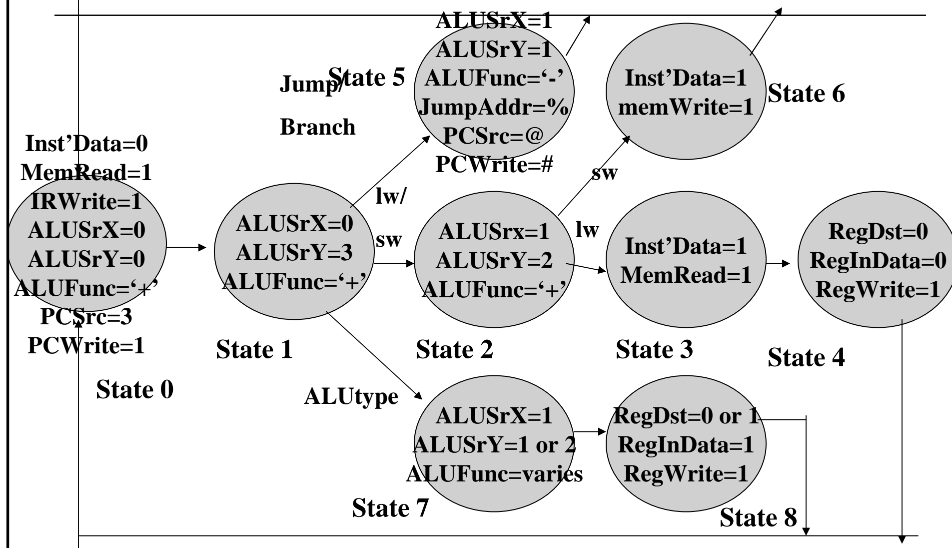
Current State (CS)

```
always @(posedge Clk or posedge reset)
begin
   if(Reset)
        currentstate=ST0;
    else
        currentstate=Nextstate;
 end
```

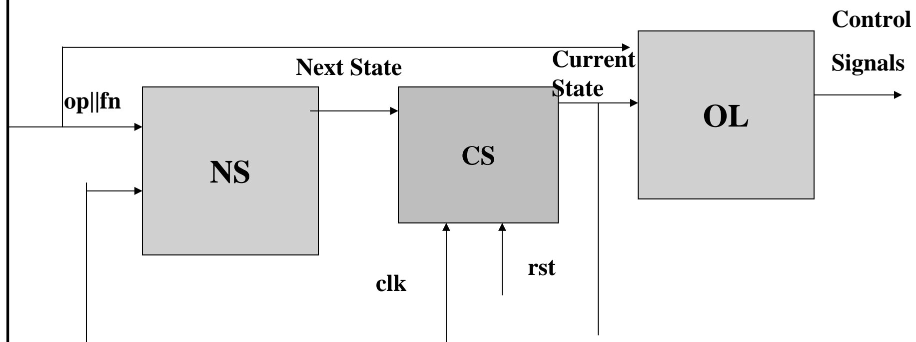
Output Logic (OL)

```
always @(Currentstate)
begin
   case(Currentstate)
       ST0: Y=1;
       ST1: Y=2;
       ST2: Y=3;
       ST3: Y=4;
end
```

The control state machine



The Controller



Performance of the Multicycle Design

- The multi-cycle implementation has a larger CPI than the single cycle implementation.
- Compute, the average CPI for:

  Rtype 44%

  Load 24%

  Store 12%

  Branch 18%

  Jump 2%

Calculating CPI

Contribution to CPI

Rtype 44%: 4 cycles => 1.76

Load 24% : 5 cycles=> 1.20

Store 12%: 4 cycles=> 0.48

Branch 18%: 3 cycles=>0.54

Jump 2%: 3 cycles=> 0.06

Thus, average CPI = 4.04

Clock frequency = 500 MHz (for 2 ns clock duration)

This, corresponds to a performance of 500/4.04=123.8 MIPS!!

Example

- Consider a MIPS++ processor, which is similar to our processor, except there are 3 types of R-type instructions:
  - R_a-type: half of all R-type instructions, 4 cycles
  - R_b-type: ¼ th of all R-type instructions, 6 cycles
  - R_c-type: ¼ th of all R-type instructions, 10 cycles
- With the same instruction mix in the last example, and assuming the slowest R-type instruction takes 16ns to execute in a single cycle implementation , derive the performance ration for a multi-cycle implementation.

Answer

- Single-cycle: 62.5 MIPS
  Multi-cycle: 101.6 MIPS
- Inclusion of more **complex type instructions**, have small effect on the CPI of a multi-cycle implementation.
- However it has a significant effect on that of a single cycle implementation.

Microprogramming

- The control state machine resembles a program that has instructions, states, branching, and loops.
- We call such a hardware program a micro-program.
- Its basic steps are called as micro-instructions.
- Within each micro-instruction, there are different actions being performed, being called as micro-order.

Micro-program vs Hardwired Controller

- Instead of implementing the controller state machine in custom hardware, we can store the micro-instructions in a ROM.
- Hence, a program is broken into machine instructions.
- A machine instruction is in turn broken into a sequence of micro instructions.
- Each micro-instruction, thus defines a step in the execution of a machine language instruction.

Advantages

- More regular.
- Less dependent on the Instruction-set architecture.
  - The same hardware can be reused by simply changing the content of the ROM.
- Errors and omissions can be taken care of by simply changing the micro-program, rather than redesigning the circuit.
- Microprogramming is designing a suitable sequence of microinstructions to realize a particular ISA.

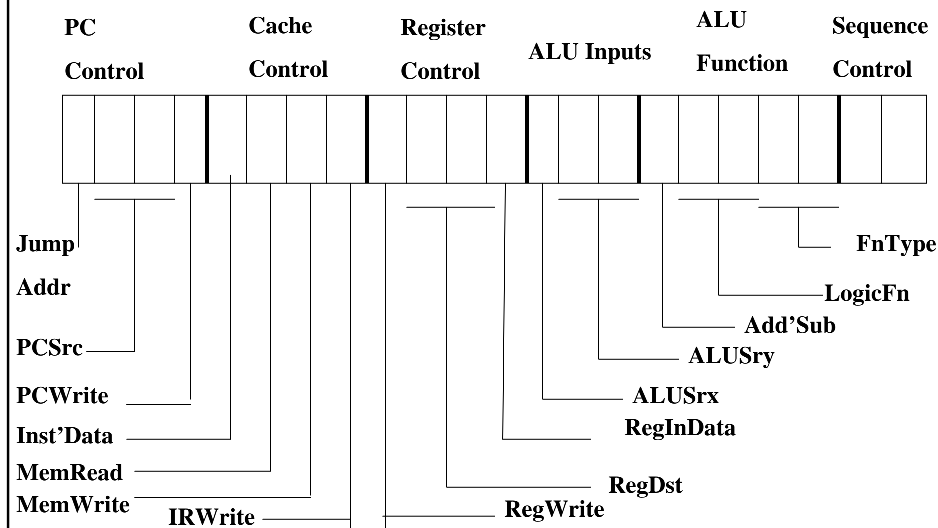
Disadvantage

- Lower speed compared to a hardwired control circuit.
- Each machine level instruction takes 3-5 ROM accesses to fetch the micro-instructions.
- After each micro-instruction has been read and placed in the micro-instruction register, sufficient time has to be given to allow the signals to stabilize and the actions to take place.

Micro-instruction format

- The design of the microcontrolled controller begins with a format.
- Each of the 20 control signals bear one-one relationship with the control bits.
- Except for the last 2 bit Sequence control signal.

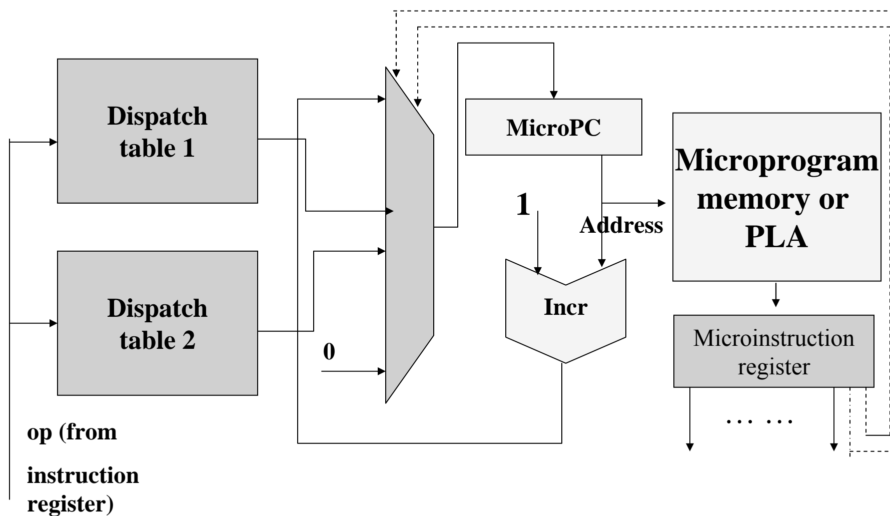
MicroMIPS instruction format



Sequence Control Bits

- The 2-bit sequence control bits allow for the control of micro-instruction sequencing in the same way that "PC control" affects the sequencing of machine language instruction.
- Option 0 is to advance to the next micro-instruction in sequence by incrementing the μPC.
- Option 1 and 2 allow branching, depending on the opcode of the instruction.
- Option 3 is to go to the microinstruction 0 corresponding to state 0; this initiates the fetch phase of the next machine instruction.

Microprogrammed control unit



Dispatch tables

- Each of the two dispatch tables translates the opcode into a microinstruction address.
- Dispatch table 1 corresponds to the multi-way branch in going from cycle 2 to 3.
- Dispatch table 2 implements the branch between cycles 3 and 4.

Microinstruction field values and their symbolic names (default value is 0)

PC control	0001 PCjump	1001 syscall	X011 PCjreg	X101 PCbranch	X111 PCnext
Cache Control	0101 Cache Fetch	1010 Cache Store	1100 Cache Load		
Register Control	1000 rt←Data	1001 rt←z	1011 rd←z	1101 $31←PC	
ALU inputs	000 PC○ 4	011 PC○ 4imm	101 x○y	110 x○imm	
ALU function	0xx10 + X1011 XOR	1xx01 < X1111 NOR	1xx10 - Xxx00 lui	X0011 Λ	X0111 V
Sequence Control	01 µPCdisp1	10 µPCdisp2	11 µPCfetch		

Micro-program

- x111 0101 0000 000 0xx10 00

is equivalent to:

PCnext, Cache Fetch, PC + 4

Complete Micro-program

fetch:	PCnext,CacheFetch, PC+4 PC+4imm,µPCdisp1	State 0 (start) State 1
lui1:	lui(imm) rt←z, µPCfetch	State 7lui State 8lui
addi:	x+y rd←z, µPCfetch	State 7add State 8add
subi:	x-y rd←z, µPCfetch	State 7sub State 8sub
slt1:	x-y rd←z, µPCfetch	State 7slt State 8slt
addi1:	x+imm rd←z, µPCfetch	State 7addi State 8addi

Complete Micro-program (Contd.)

slti1:	x-imm rt←z, μPCfetch	State 7slti State 8slti
and1:	xΛy rd←z, μPCfetch	State 7and State 8and
or1:	xVy rd←z, μPCfetch	State 7add State 8add
xor1:	xVy rd←z, μPCfetch	State 7or State 8or
nor1:	x~Vy rd←z, μPCfetch	State 7nor State 8nor
andi1:	xΛimm rt←z, μPCfetch	State 7andi State 8andi

Complete Micro-program (Contd.)

ori1:	xVimm rt←z, μPCfetch	State 7ori State 8ori
xori1:	xΦimm rd←z, μPCfetch	State 7xori State 8xori
lwsw1:	x+imm, μPCdisp2	State 2
lw2:	CacheLoad rd←Data, μPCfetch	State 3 State 4
sw2:	CacheStore, μPCfetch	State 6

Complete Micro-program (Contd.)

j1:	PCjump, µPCfetch	State 5j
jr1:	PCjreg, µPCfetch	State 5jr
branch1:	PCbranch, µPCfetch	State 5branch
jal1:	PCjump, \$31←PC, µPCfetch	State 5jal
syscall:	PCsyscall, µPCfetch	State 5syscall

Comments

- Each line represents micro-instructions.
- The label 1(2) is to indicate that they are arrived from dispatch table 1(2).
- The top-most microinstruction (fetch) is stored at ROM address 0.
- Thus starting the machine with µPC cleared to 0, will cause program execution to start from location 0.

Assignment (not for submission)

Simplify the micro-instruction format, and design the micro-programs for the ISA, if the 5 ALU bits are directly generated in a separate decoder and fed to the ALU.

Horizontal vs Vertical Microinstruction

- The instruction discussed with separate bits for each of the 20 control bits of the datapath is called horizontal microinstruction.
- However, suitable encoding can reduce the size of the instructions.
  - Eg. the cache control field has four values, which can be encoded in 2 bits.
- Such an encoded instruction format is called as vertical microinstruction.
- However, they get slower as they need further decoders.