TSEA26 Tutorial 4. Design of Program Control Flow Units

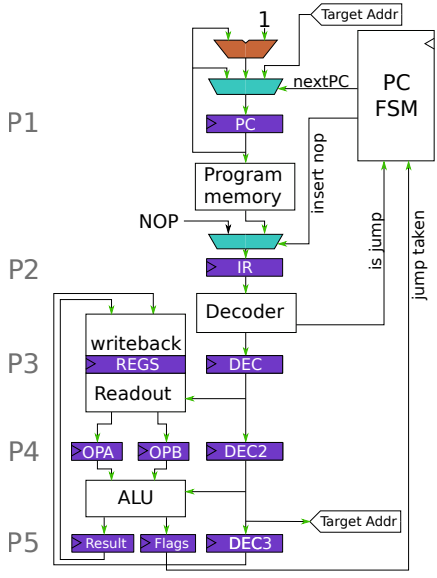
Frans Skarman [1]

November 29, 2021

[1] based on material by Oscar Gustafsson, Dake Liu, Olle Seger, Andreas Ehliar, and Jian Wang

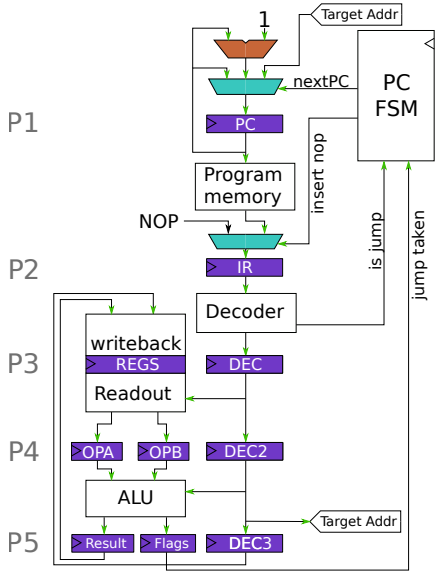
Program Flow Control Unit

- Choose which instruction to execute next
- Easy in theory, harder in practice
  - Hardware repeat
  - Conditional jumps
  - Call/return
  - ...
- PFC needs to
  - Flush pipeline
  - Insert nops
  - Handle delay slots
  - ...



A pipelined CPU

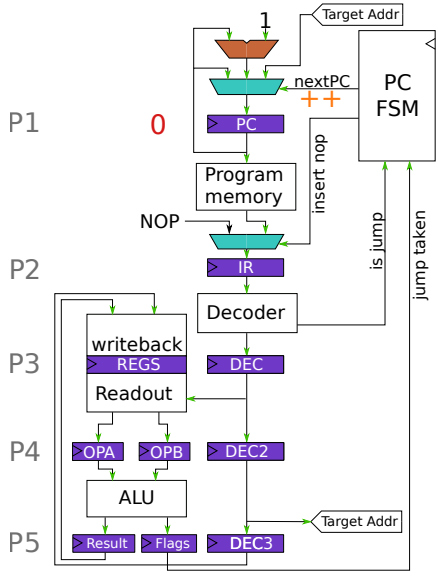
- Similar to senior
- PC FSM controls behaviour during jumps
  - When to insert NOPs
  - When to Stall
- Inputs:
  - **is jump** is next insn. a jump? If so which kind?
  - **jump taken** should this conditional jump be executed?
- Outputs:
  - **insert nop** replace current insn. with nop?
  - **nextPC** next PC = target address, ++ or stall



PC fsm example

Try running through the following program

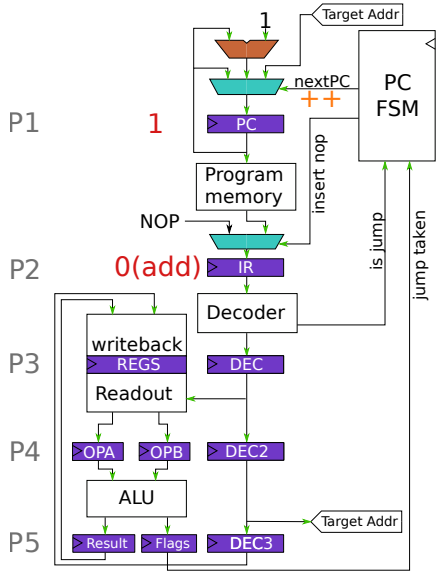
```
pc
0:   add r0, r1, r2
1:   jump.eq 9
2:   xxx
```



PC fsm example

Try running through the following program

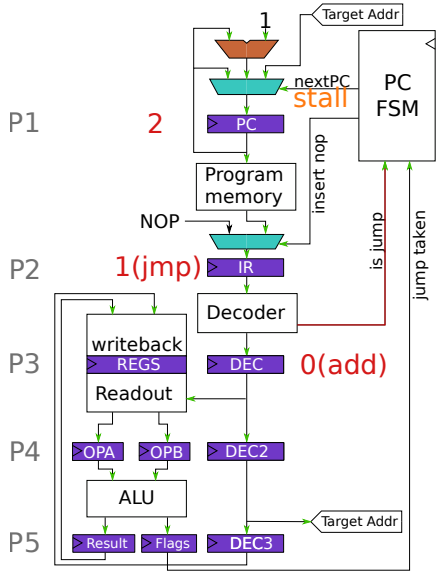
```
pc
0:   add r0, r1, r2
1:   jump.eq 9
2:   xxx
```



PC fsm example

Try running through the following program

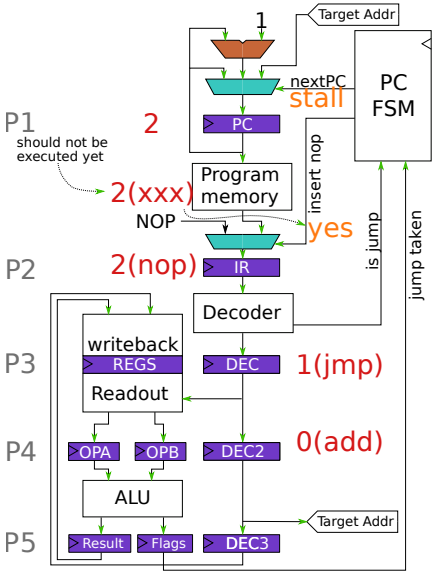
```
pc
0:   add r0, r1, r2
1:   jump.eq 9
2:   xxx
```



PC fsm example

Try running through the following program

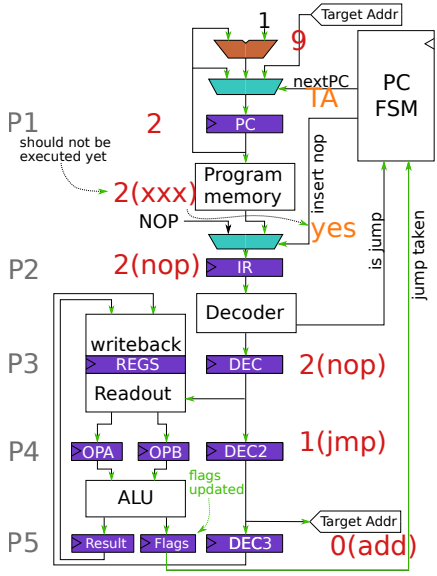
```
pc
0:   add r0, r1, r2
1:   jump.eq 9
2:   xxx
```



PC fsm example

Try running through the following program

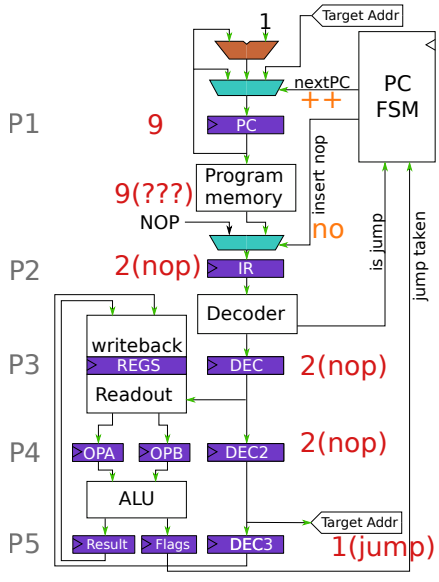
```
pc
0:   add r0, r1, r2
1:   jump.eq 9
2:   xxx
```



PC fsm example

Try running through the following program

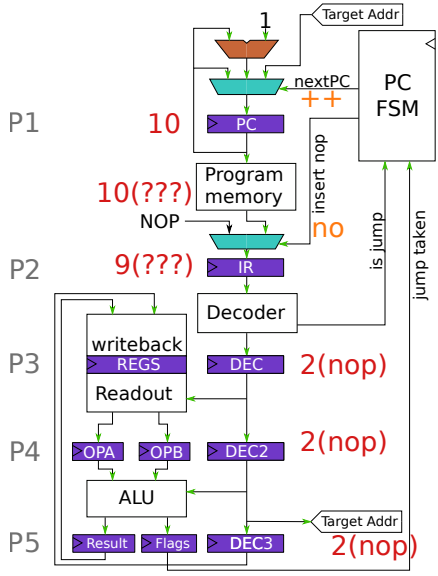
```
pc
0:   add r0, r1, r2
1:   jump.eq 9
2:   xxx
```



PC fsm example

Try running through the following program

```
pc
0:   add r0, r1, r2
1:   jump.eq 9
2:   xxx
```

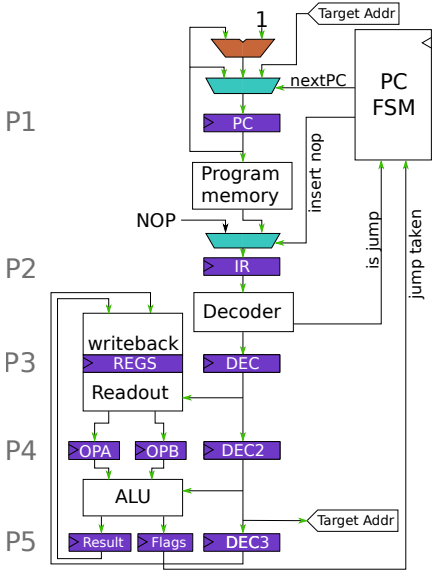


Pipeline table

- Helpful tool to figure out when and where to set PFC signals
- Each column represents a pipeline stage or a signal

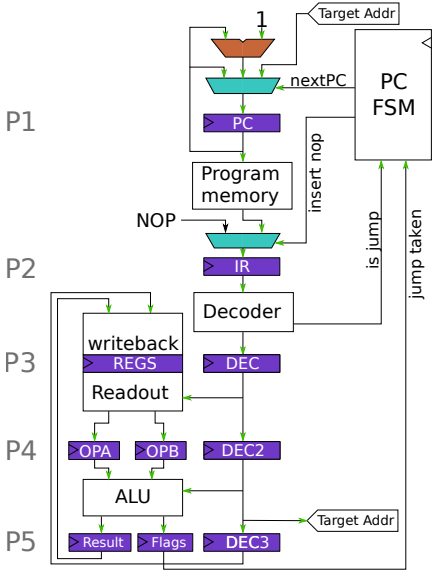
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp		0(add)			
	2	xxx		1(jmp)	0(add)		
	2	xxx		nop	1(jmp)	0(add)	
	2	xxx		nop	nop	1(jmp)	0(add)
++	9	9(aaa)	no	nop	nop	nop	1(jmp)
++	10	10(bbb)	no	9(bbb)	nop	nop	nop

Pipeline table



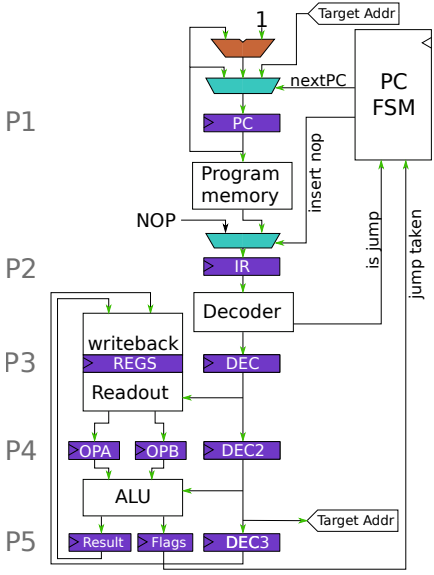
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				

Pipeline table



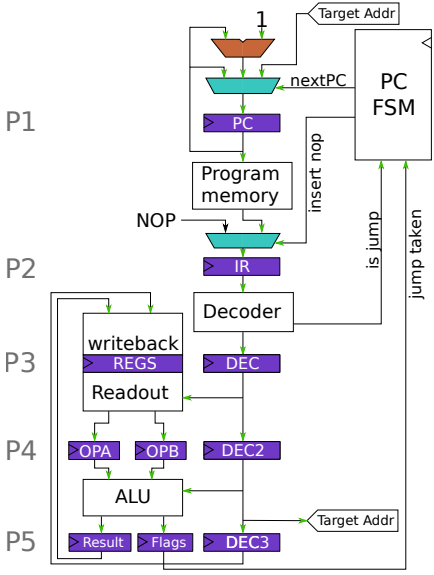
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			

Pipeline table



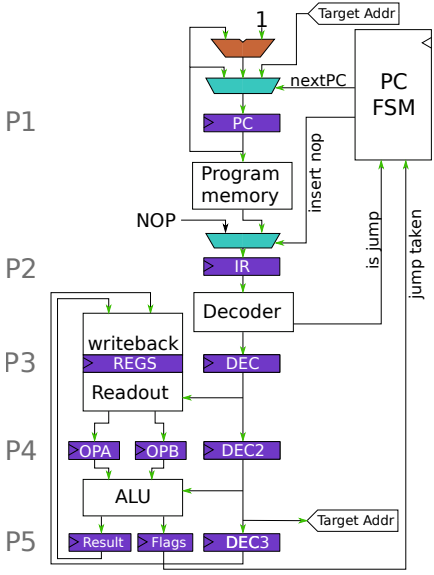
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			

Pipeline table



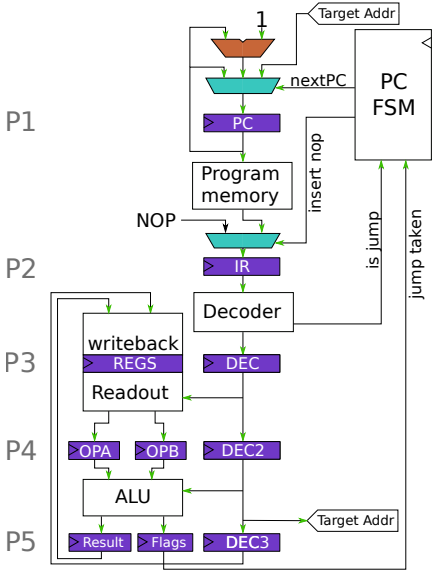
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
	2	xxx		1(jmp)	0(add)		

Pipeline table



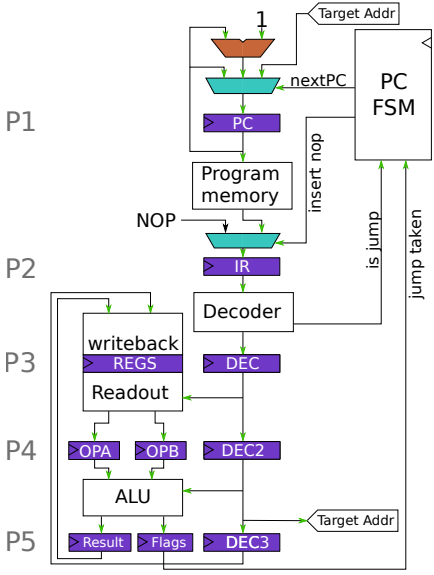
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
stall	2	xxx		1(jmp)	0(add)		

Pipeline table



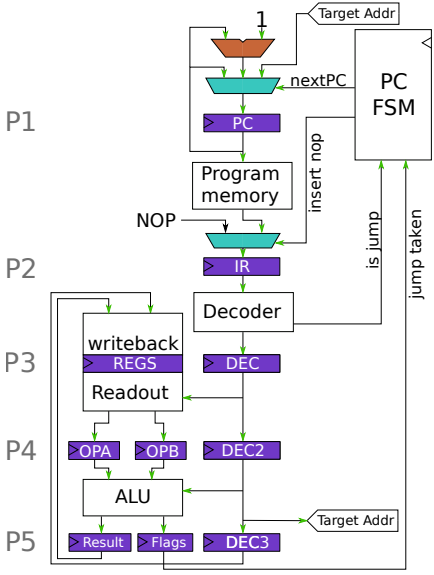
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
stall	2	xxx	yes	1(jmp)	0(add)		

Pipeline table



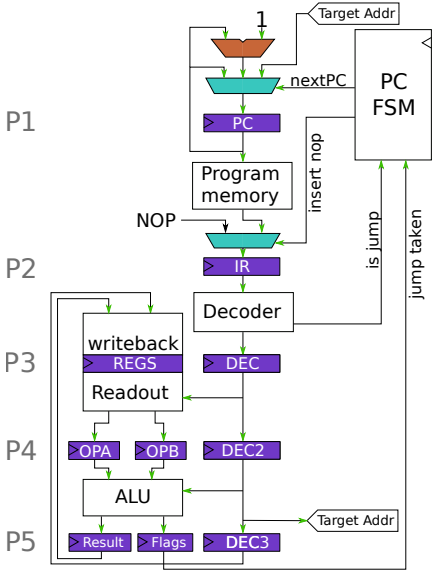
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
stall	2	xxx	yes	1(jmp)	0(add)		
	2	xxx		nop	1(jmp)	0(add)	

Pipeline table



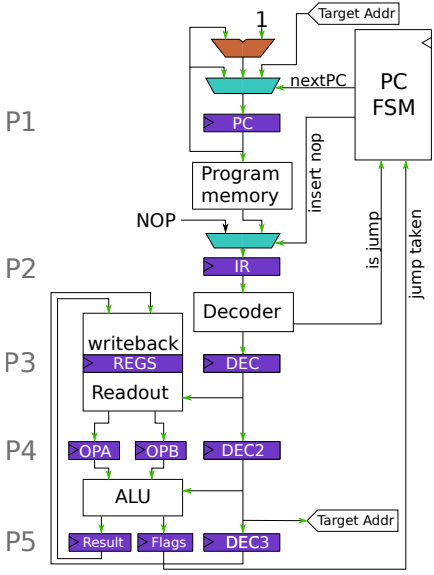
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
stall	2	xxx	yes	1(jmp)	0(add)		
stall	2	xxx		nop	1(jmp)	0(add)	

Pipeline table



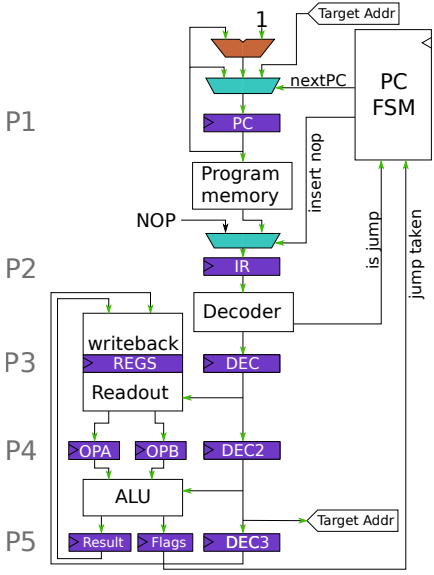
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
stall	2	xxx	yes	1(jmp)	0(add)		
stall	2	xxx	yes	nop	1(jmp)	0(add)	

Pipeline table



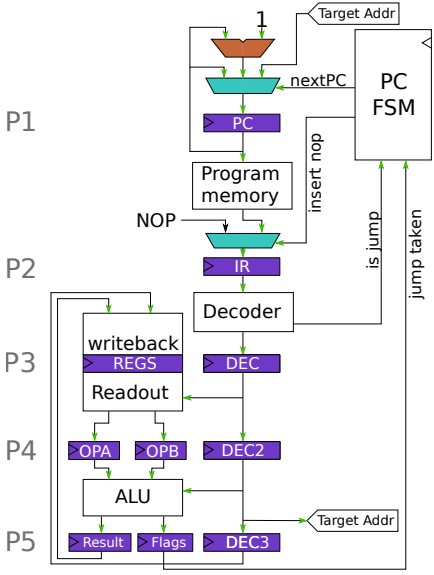
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
stall	2	xxx	yes	1(jmp)	0(add)		
stall	2	xxx	yes	nop	1(jmp)	0(add)	
	2	xxx		nop	nop	1(jmp)	0(add)

Pipeline table



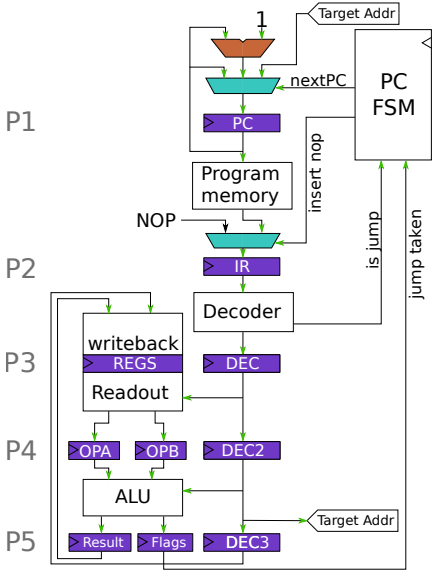
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
stall	2	xxx	yes	1(jmp)	0(add)		
stall	2	xxx	yes	nop	1(jmp)	0(add)	
TA	2	xxx		nop	nop	1(jmp)	0(add)

Pipeline table



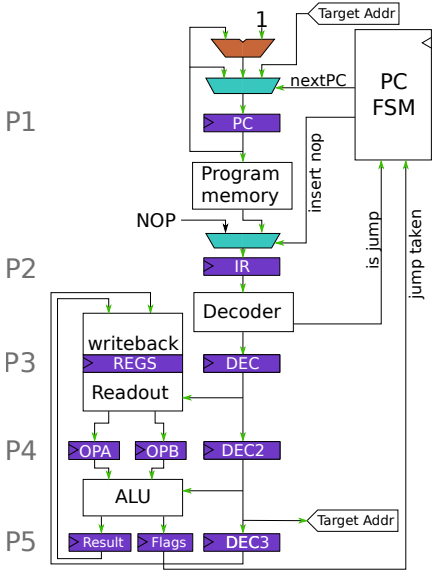
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
stall	2	xxx	yes	1(jmp)	0(add)		
stall	2	xxx	yes	nop	1(jmp)	0(add)	
TA	2	xxx	yes	nop	nop	1(jmp)	0(add)

Pipeline table



nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
stall	2	xxx	yes	1(jmp)	0(add)		
stall	2	xxx	yes	nop	1(jmp)	0(add)	
TA	2	xxx	yes	nop	nop	1(jmp)	0(add)
++	9	9(aaa)	no	nop	nop	nop	1(jmp)

Pipeline table



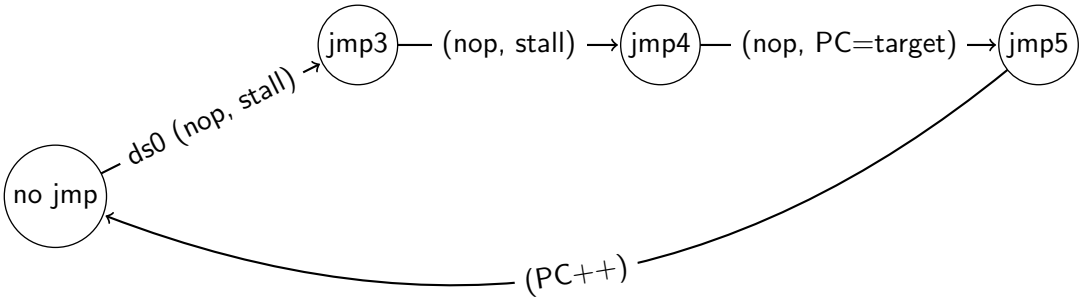
nextPC	**P1**	*PM*	*nop?*	**P2**	**P3**	**P4**	**P5**
++	0	add	no				
++	1	jmp	no	0(add)			
stall	2	xxx	yes	1(jmp)	0(add)		
stall	2	xxx	yes	nop	1(jmp)	0(add)	
TA	2	xxx	yes	nop	nop	1(jmp)	0(add)
++	9	9(aaa)	no	nop	nop	nop	1(jmp)
++	10	10(bbb)	no	9(bbb)	nop	nop	nop

Finite state machines

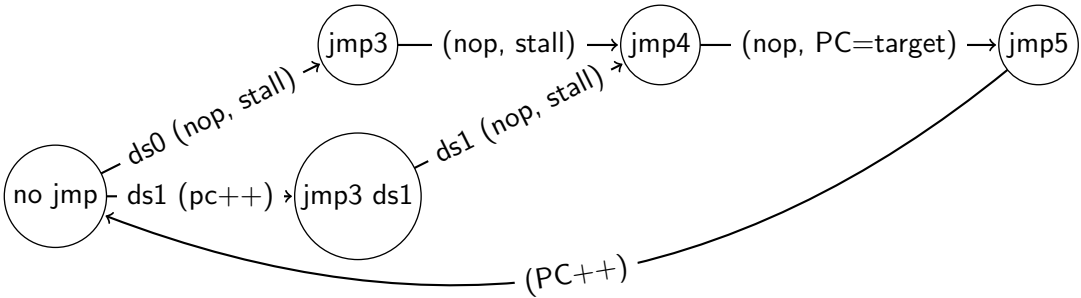
Used in the PCF unit to set the appropriate control signals. State depends on where jump is in pipeline and delay slots

- Can be Moore
  - Output only depends on current state
  - Output is delayed 1 cycle
- Or Mealy
  - Output is function of state and input
  - Can react instantly
  - But you can create combinatorial loops
- We want quick reaction to jump instructions
- $\Rightarrow$ Mealy

PC FSM State graph



With delay slots



Speculative execution

- Jumps have a lot of overhead, 3 cycles from start to finish
- Speculative execution can help
- Branch prediction
  - "Guess" the outcome of a branch
  - Statically try to predict result based on heuristics
  - Dynamically try to predict result based on history
  - ...
- Branch target prediction
  - Guess the branch of the target before it's known/calculated
- Return address buffering
  - Keep a small stack of return addresses in registers
  - And the rest in main memory
- Be weary of security issues (spectre, meltdown)
- Not part of the course

Exercises

- Exercise 4.1 and 4.3