Computer Architecture Experiment

Topic 4. Pipelined CPU with Cache

浙江大学计算机学院

2022年10月

Outline

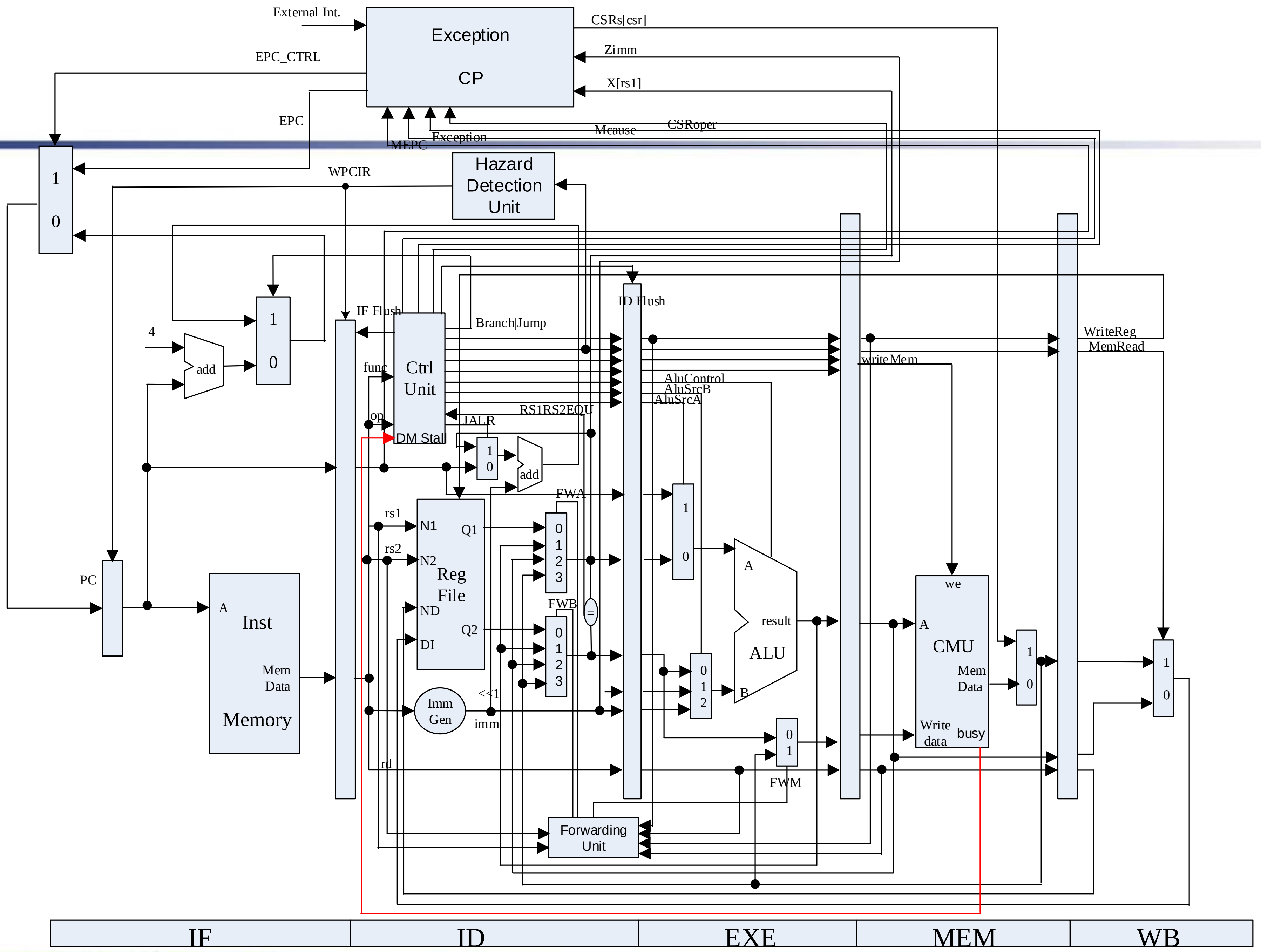
- Experiment Purpose
- Experiment Task
- Basic Principle
- Operating Procedures
- Checkpoints

Experiment Purpose

- **Understand the principle of Cache Management Unit (CMU) and State Machine of CMU**
- **Master the design methods of CMU and Integrate it to the CPU.**
- **Master verification methods of CMU and compare the performance of CPU when it has cache or not.**

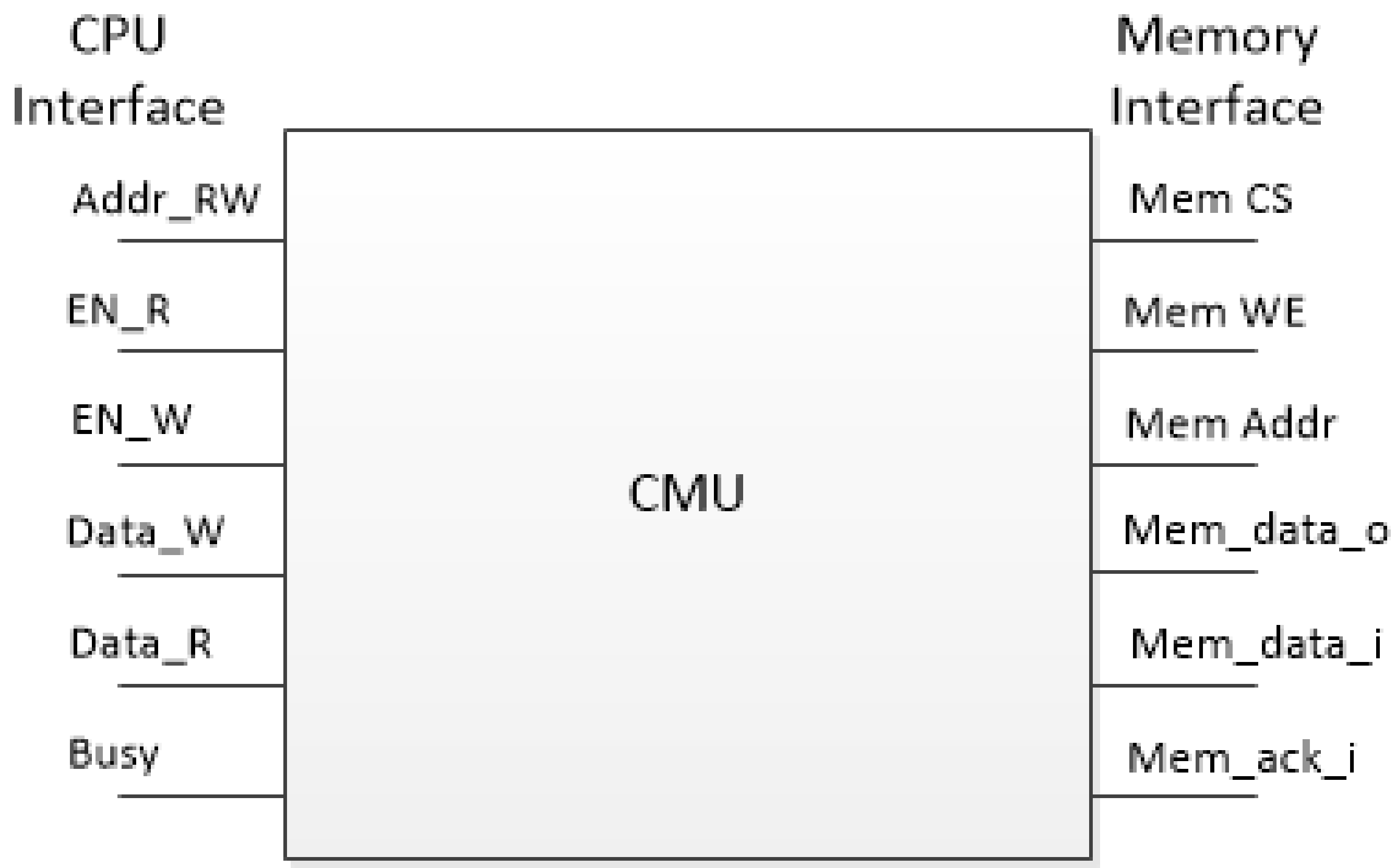
Experiment Task

- Design of Cache Management Unit and integrate it to CPU.
- Observe and Analyze the Waveform of Simulation.
- Compare the performance of CPU when it has cache or not.



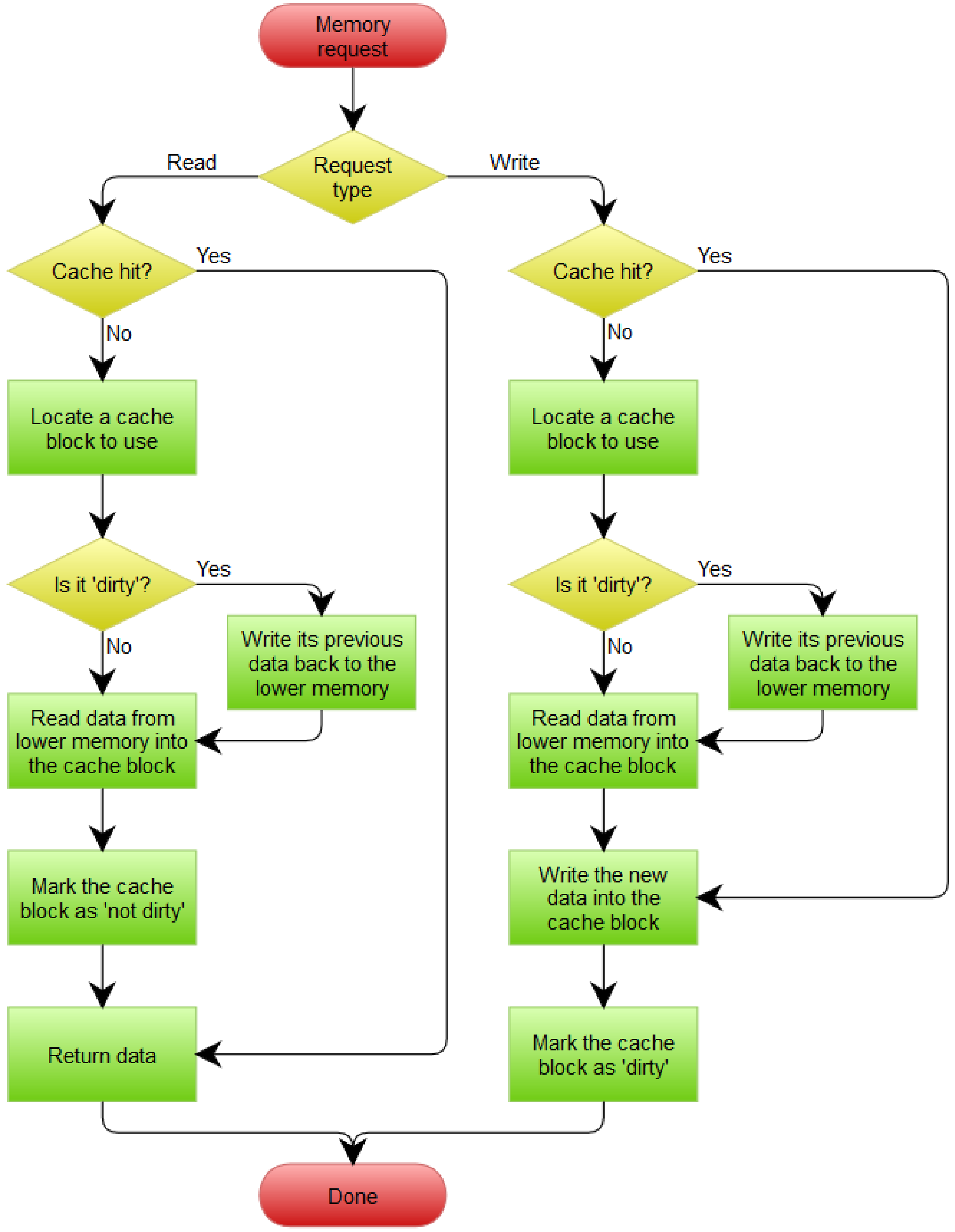
External Int.
Exception
CP
CSRs[csr]
Zimm
X[rs1]
EPC_CTRL
EPC
MEPC
Exception
Mcause
CSRoper
Hazard Detection Unit
WPCIR
1
0
4
add
IF Flush
Ctrl Unit
func
op
DM Stall
Branch|Jump
ID Flush
JALR
RS1RS2EQU
AluControl
AluSrcB
AluSrcA
writeMem
WriteReg
MemRead
PC
A
Inst Memory
Mem Data
rs1
rs2
rd
N1
N2
ND
DI
Q1
Q2
Reg File
Imm Gen
imm
<<1
FWA
FWB
FWM
0 1 2 3
ALU
A
B
result
we
A
CMU
Mem Data
Write data
busy
Forwarding Unit
IF
ID
EXE
MEM
WB

Cache Management Unit

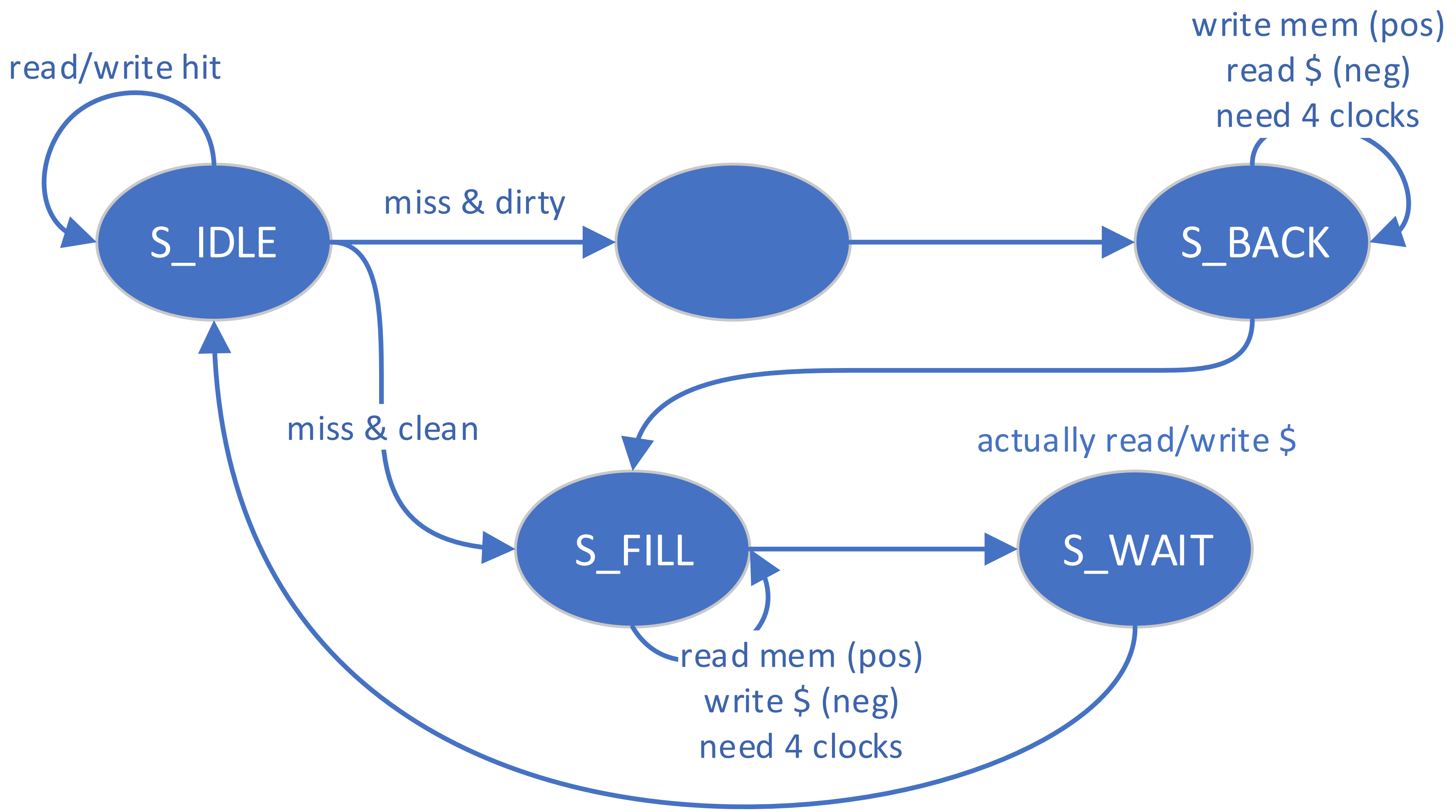


Cache Operation Flc

- **Read (Hit/Miss)**
- **Write (Hit/Miss)**
- **Replace (Clean/Dirty)**

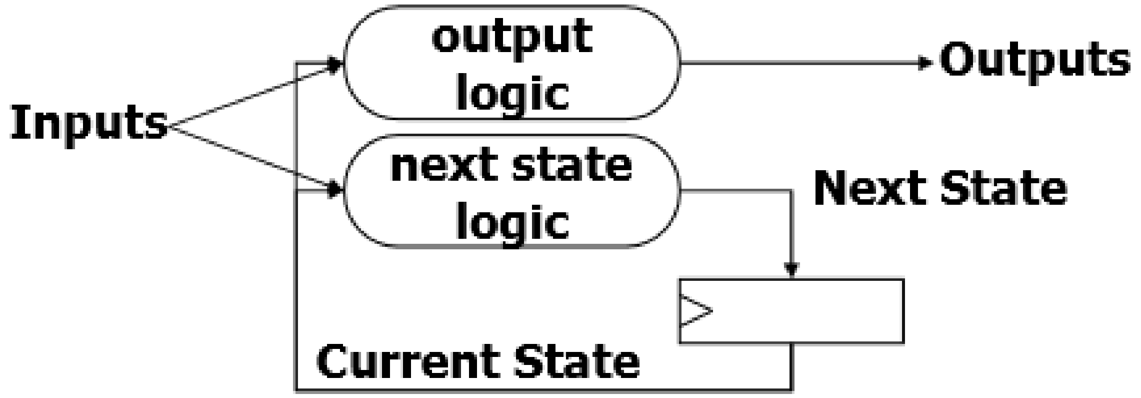

Memory request
Request type
Read
Write
Cache hit?
Yes
No
Locate a cache block to use
Is it 'dirty'?
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Mark the cache block as 'not dirty'
Return data
Cache hit?
Yes
No
Locate a cache block to use
Is it 'dirty'?
Yes
No
Write its previous data back to the lower memory
Read data from lower memory into the cache block
Write the new data into the cache block
Mark the cache block as 'dirty'
Done

Cache Management State Machine



State Machine

- Next State Logic
- State assignment
- Output



RV32 Core

```
module cmu (
                    input clk,                              // CPU side
                    input rst,
                    input [31:0] addr_rw,
                    input en_r,
                    input en_w,
                    input [2:0] u_b_h_w,
                    input [31:0] data_w,
                    output [31:0] data_r,
                    output stall,
                    output reg mem_cs_o = 0,                // mem side
                    output reg mem_we_o = 0,
                    output reg [31:0] mem_addr_o = 0,
                    input [31:0] mem_data_i,
                    output [31:0] mem_data_o,
                    input mem_ack_i,
                    output [2:0] cmu_state                  // debug info
          );
```

Instr. Mem.(1)

NO.	Instruction	Addr.	Label	ASM	Comment
0	00000013	0	__start:	addi x0, x0, 0	
1	01c00083	4		lb x1, 0x01C(x0)	# F0F0F0F0 in 0x1C # FFFFFFF0 miss, read 0x010~0x01F to set 1 line 0
2	01c01103	8		lh x2, 0x01C(x0)	# FFFFF0F0 hit cycle 5~22
3	01c02183	C		lw x3, 0x01C(x0)	# F0F0F0F0 hit
4	01c04203	10		lbu x4, 0x01C(x0)	# 000000F0 hit
5	01c05283	14		lhu x5, 0x01C(x0)	# 0000F0F0 hit
6	21002003	18		lw x0, 0x210(x0)	# miss, read 0x210~0x21F to cache set 1 line 1
7	abcde0b7	1C		lui x1 0xABCDE	# x1 = 0xABCDE000 cycle 27~44
8	402200b3	20		sub x1, x4, x2	# x1 = 0x0000_1000
9	71c08093	24		addi x1, x1, 0x71C	# x1 = 0x0000171C
10	00100023	28		sb x1, 0x0(x0)	# miss, read 0x000~0x00F to cache set 0 line 0
11	00101223	2C		sh x1, 0x4(x0)	# hit
12	00102423	30		sw x1, 0x8(x0)	# hit

Instr. Mem.(2)

NO.	Instruction	Addr.	Label	ASM	Comment
13	20002303	34		lw x6, 0x200(x0)	# miss, read 0x200~0x20F to cache set 0 line 1
14	40002383	38		lw x7, 0x400(x0)	# miss, write 0x000~0x00F back to ram, then read 0x400~40F to cache set 0 line 0
15	41002403	3C		lw x8, 0x410(x0)	# miss, no write back because of clean, read 0x410~41F to cache set 1 line 0
16	0ed06813	40	loop:	ori x16, x0, 0xED	# end
17	ffdff06f	44		jal x0, loop	

Data Mem.

NO.	Data	Addr.	Comment
0	000080BF	0	
1	00000008	4	
2	00000010	8	
3	00000014	C	
4	FFFF0000	10	
5	0FFF0000	14	
6	FF000F0F	18	
7	F0F0F0F0	1C	
8	00000000	20	
9	00000000	24	
10	00000000	28	
11	00000000	2C	
12	00000000	30	
13	00000000	34	
14	00000000	38	
15	00000000	3C	

NO.	Instruction	Addr.	Comment
16	00000000	40	
17	00000000	44	
18	00000000	48	
19	00000000	4C	
20	A3000000	50	
21	27000000	54	
22	79000000	58	
23	15100000	5C	
24	00000000	60	
25	00000000	64	
26	00000000	68	
27	00000000	6C	
28	00000000	70	
29	00000000	74	
30	00000000	78	
31	00000000	7C	

Checkpoints

- **CP1:**

  Waveform Simulation of CMU.

- **CP2:**

  FPGA Verification.

Tips

- **对于 reg_MW 流水线寄存器：当出现memory stall之后，WB阶段的指令只能执行一个cycle，之后被flush掉**
- **原因**
  - WB阶段执行完毕之后必须flush，否则不符合流水化
  - 在目前的单条流水线上，如果不flush，也无所谓，会一直对某个寄存器不断写相同的数据
  - 但是如果是多周期流水线，不同的function unit可能都需要写，memory通路阻塞，但是别的线路可以正常写操作
  - 如果一直不flush,不仅会占据写口，还有可能会造成错误
- **建议**
  - 取消MW寄存器的EN信号，加上一个flush信号
  - （MW寄存器永远都不能hold，要么流水，要么flush）

Tips

- Data_ram 模块中也设置了状态机，目的是为了模拟真实内存读写数据需要消耗不止一个CPU cycle
- 在本实验中，通过状态机，把一次RAM的读/写操作放大到四个个CPU时钟周期
- 一个block的换入换出需要四次RAM读写操作
  - 如果分配一个clean的块：本质上需要16个CPU-cycle进行cache块的分配
  - 如果分配一个dirty的块：本质上需要32个CPU-cycle进行cache块的分配
-

Thanks!