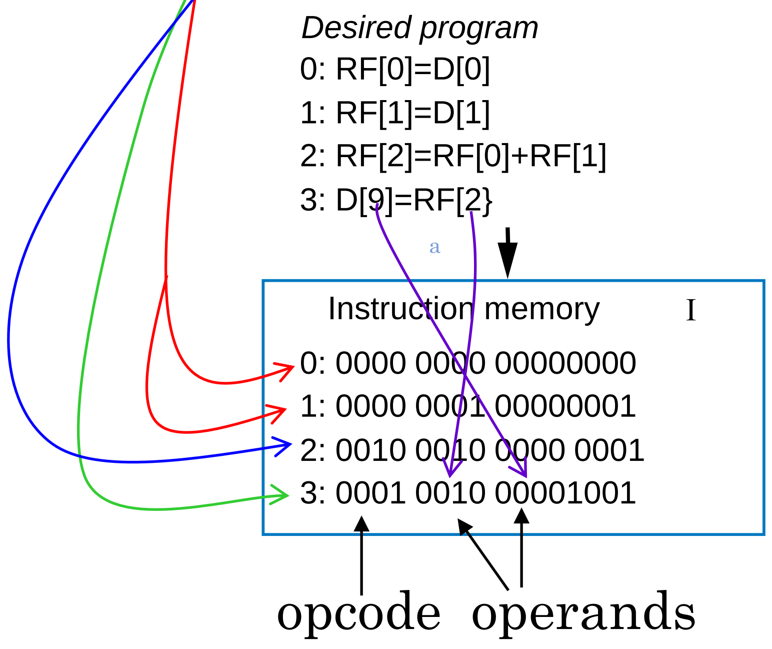
SIP 109-1 期末專題：AN EIGHT-INSTRUCTION PROGRAMMABLE PROCESSOR

Three-Instruction Programmable Processor

- Instruction Set – List of allowable instructions and their representation in memory, e.g.,
  - *Load* instruction—0000 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$
  - *Store* instruction—0001 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$
  - *Add* instruction—0010 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$



Desired program

```
0: RF[0]=D[0]
1: RF[1]=D[1]
2: RF[2]=RF[0]+RF[1]
3: D[9]=RF[2}
```

Instruction memory

```
0: 0000 0000 00000000
1: 0000 0001 00000001
2: 0010 0010 0000 0001
3: 0001 0010 00001001
```

Instructions in 0s and 1s – *machine code*

A SIX-INSTRUCTION PROGRAMMABLE PROCESSOR

- Let's add three more instructions:
  - *Load-constant* instruction—0011 $r_3r_2r_1r_0$ $c_7c_6c_5c_4c_3c_2c_1c_0$
    - MOV Ra, #c—specifies the operation *RF[a]=c*
  - *Subtract* instruction—0100 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$
    - SUB Ra, Rb, Rc—specifies the operation *RF[a]=RF[b] – RF[c]*
  - *Jump-if-zero* instruction—0101 $ra_3ra_2ra_1ra_0$ $o_7o_6o_5o_4o_3o_2o_1o_0$
    - JMPZ Ra, offset—specifies the operation *PC = PC + offset* if *RF[a]* is 0

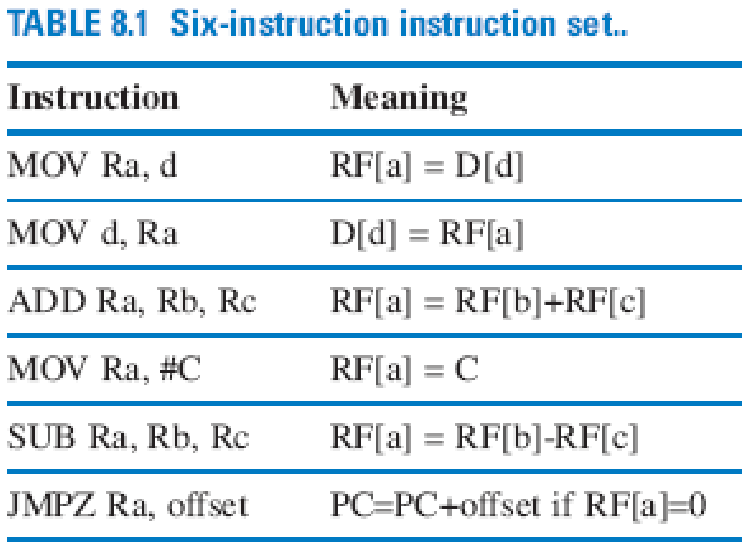


TABLE 8.1 Six-instruction instruction set..

Instruction	Meaning
MOV Ra, d	RF[a] = D[d]
MOV d, Ra	D[d] = RF[a]
ADD Ra, Rb, Rc	RF[a] = RF[b]+RF[c]
MOV Ra, #C	RF[a] = C
SUB Ra, Rb, Rc	RF[a] = RF[b]-RF[c]
JMPZ Ra, offset	PC=PC+offset if RF[a]=0

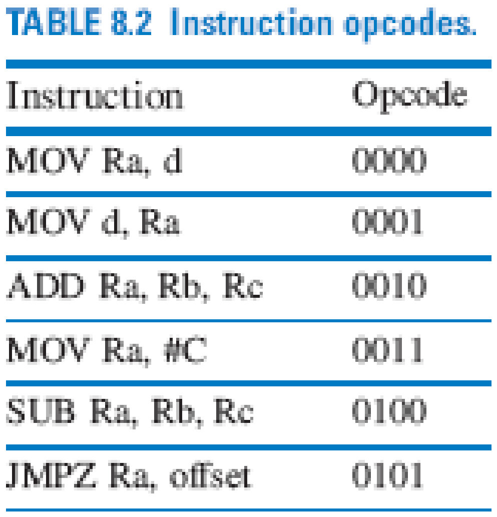


TABLE 8.2 Instruction opcodes.

Instruction	Opcode
MOV Ra, d	0000
MOV d, Ra	0001
ADD Ra, Rb, Rc	0010
MOV Ra, #C	0011
SUB Ra, Rb, Rc	0100
JMPZ Ra, offset	0101

Extending the Control-Unit and Datapath

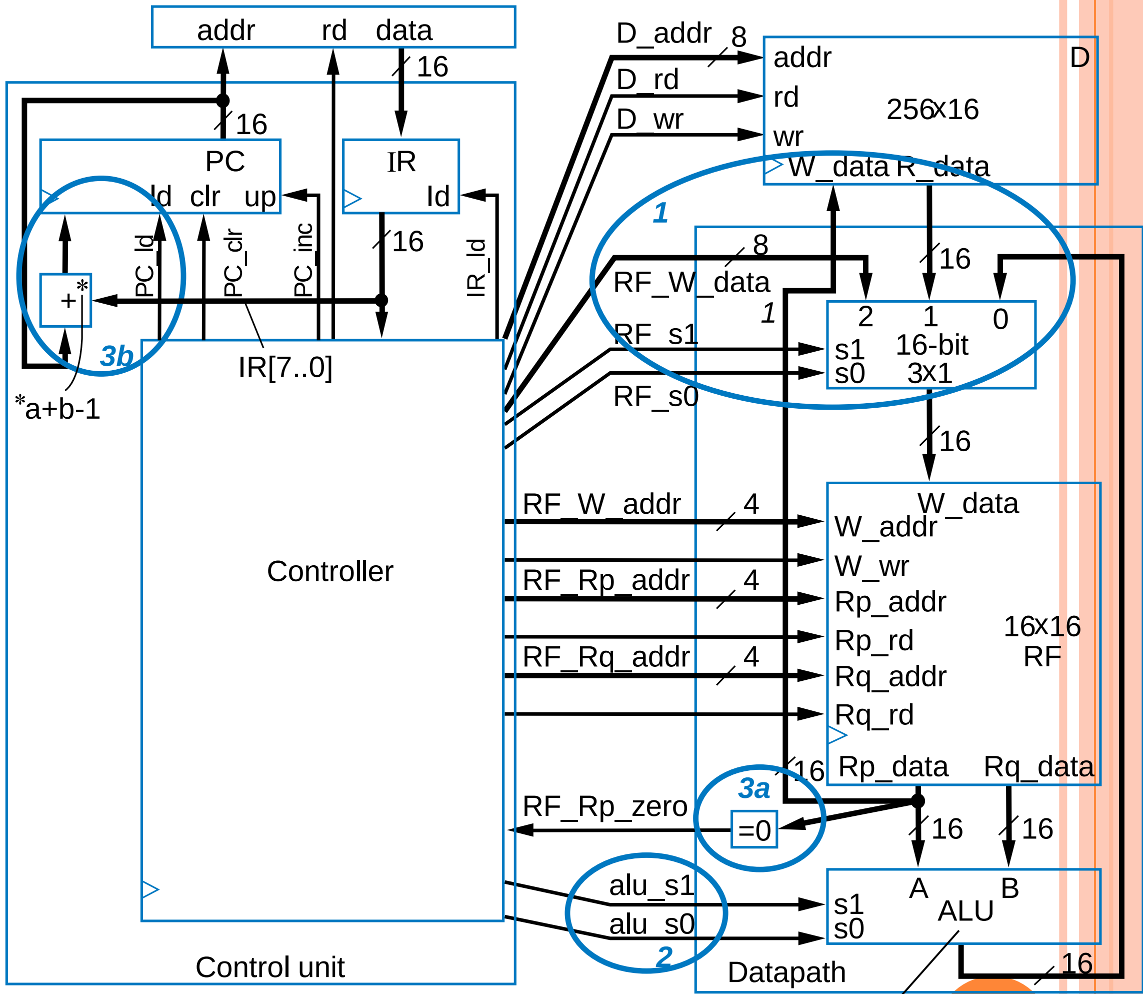
1: The *load constant* instruction requires that the register file be able to load data from *IR[7..0]*, in addition to data from data memory or the ALU output. Thus, we widen the register file's multiplexer from 2x1 to 3x1, add another mux control signal, and also create a new signal coming from the controller labeled *RF_W_data*, which will connect with *IR[7..0]*.

2: The subtract instruction requires that we use an ALU capable of subtraction, so we add another ALU control signal.

3: The jump-if-zero instruction requires that we be able to detect if a register is zero, and that we be able to add *IR[7..0]* to the *PC*.

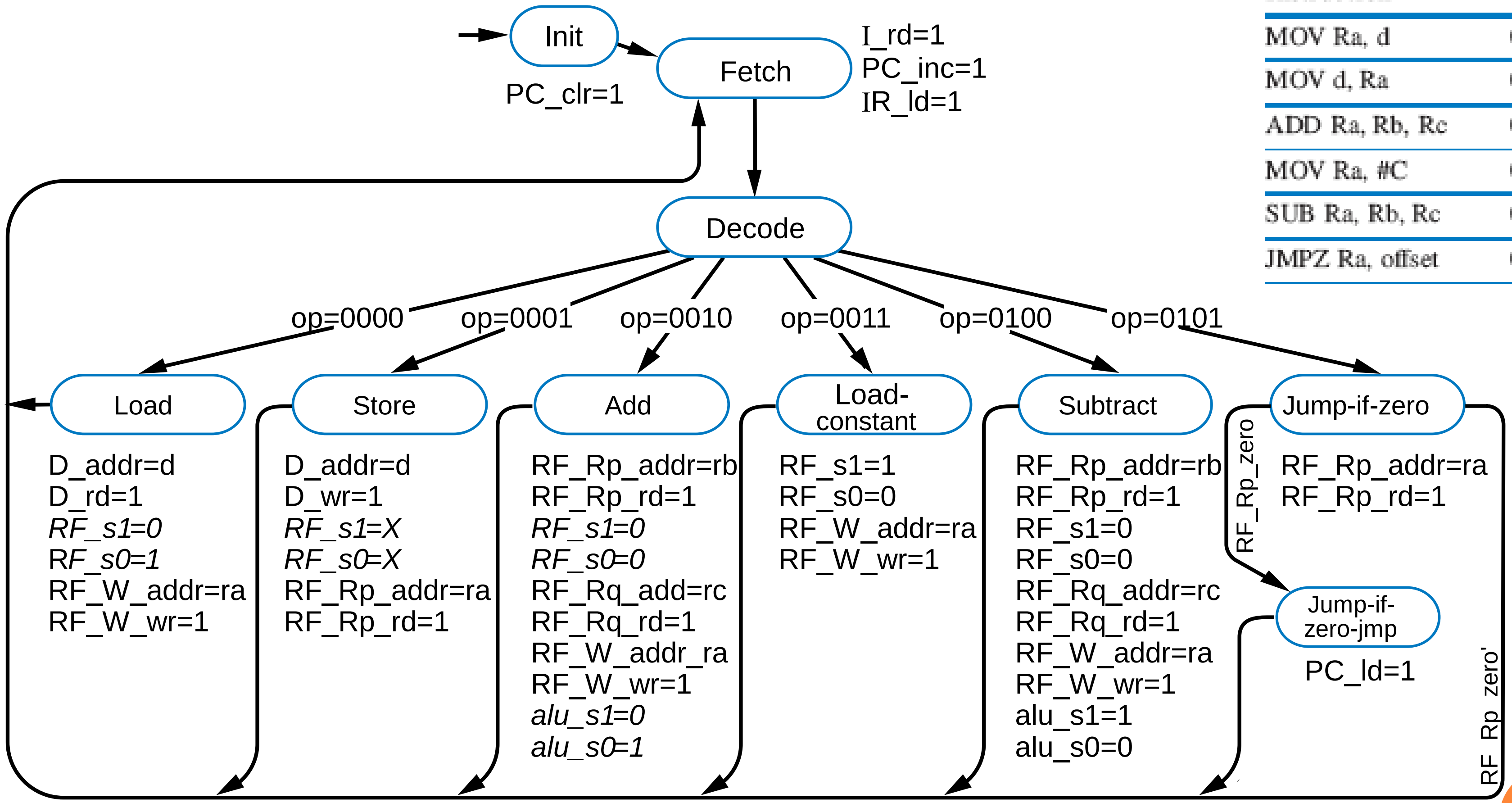
3a: We insert a datapath component to detect if the register file's *Rp* read port is all zeros (that component would just be a NOR gate).

3b: We also upgrade the *PC* register so it can be loaded with *PC* plus *IR[7..0]*. The adder used for this also subtracts 1 from the sum, to compensate for the fact that the *Fetch* state already added 1 to the *PC*.



s1	s0	ALU operation
0	0	pass A through
0	1	A+B
1	0	A-B

Controller FSM for the Six-Instruction Processor



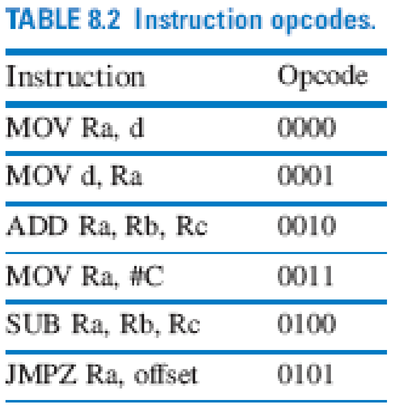
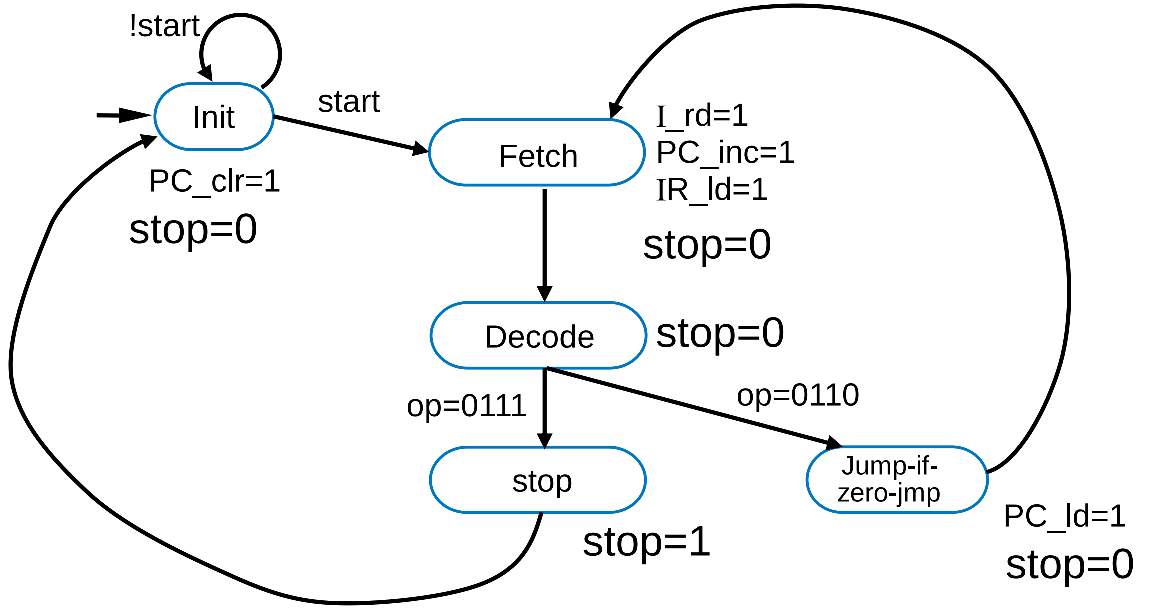


TABLE 8.2 Instruction opcodes.

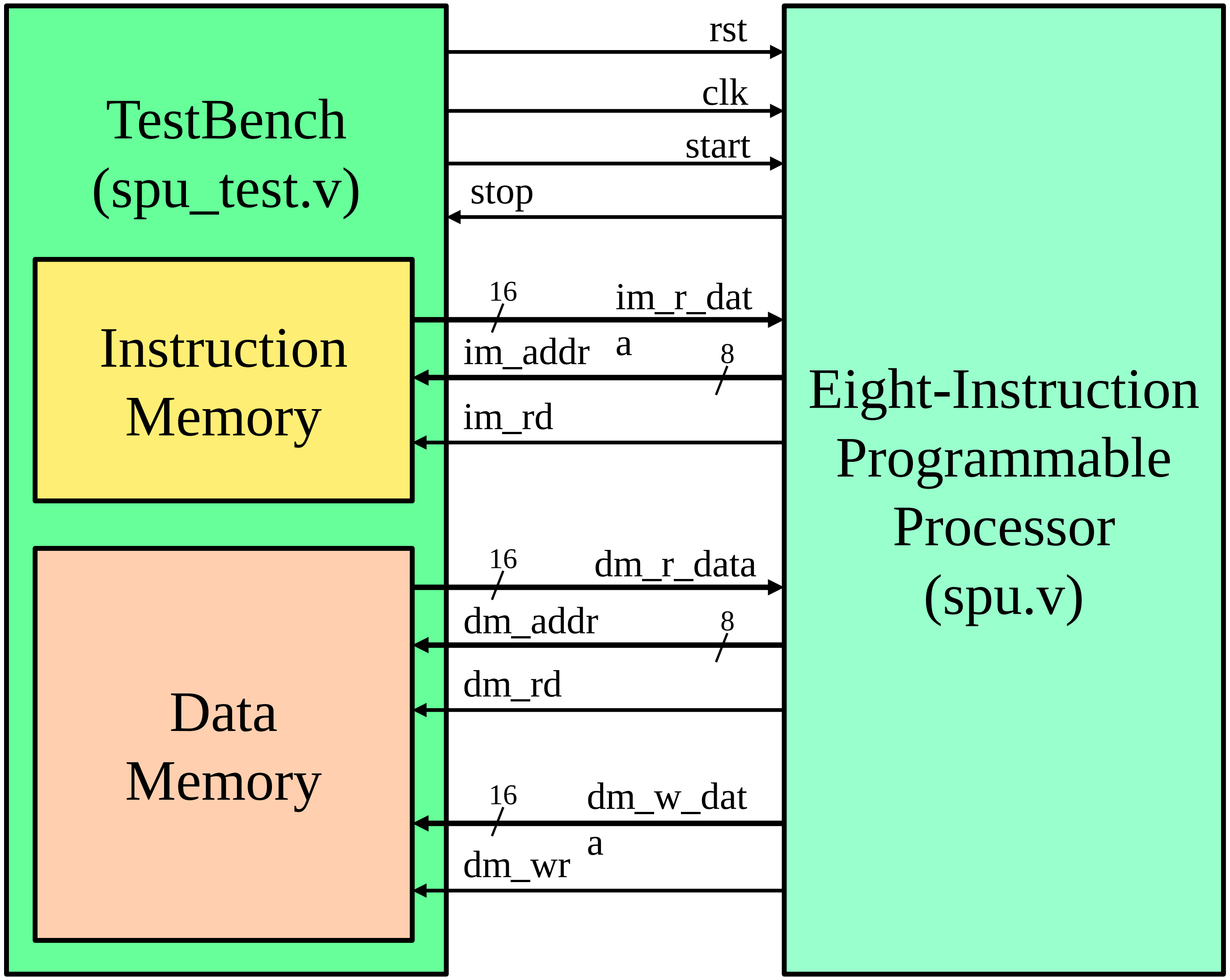
Instruction	Opcode
MOV Ra, d	0000
MOV d, Ra	0001
ADD Ra, Rb, Rc	0010
MOV Ra, #C	0011
SUB Ra, Rb, Rc	0100
JMPZ Ra, offset	0101

Increase Two more Instructions for the Programmable Processor

- Let's add two more instructions:
  - *Jump* instruction—0110 0000 $o_7o_6o_5o_4o_3o_2o_1o_0$
    - JMP offset —specifies the operation *PC = PC + offset*
  - *Stop* instruction—0111 0000 0000 0000
    - Stop —specifies the operation : stop the processor and reset
- Let's add a start input signal stop output signal to your processor:



Interface of SPU (System Block)



Interface of SPU (Table list)(1/2)

Signal name	I/O	Width	Simple Description
rst	input	1	Asynchronous reset (active high)
clk	input	1	System clock (positive edge)
start	input	1	This signal is to start the processor. (active high)
stop	output	1	This signal represents that the processor has stopped. (active high)
im_r_data	input	16	16-bit read data of instruction memory
im_addr	output	8	8-bit data address of instruction memory
im_rd	output	1	read enable of instruction memory

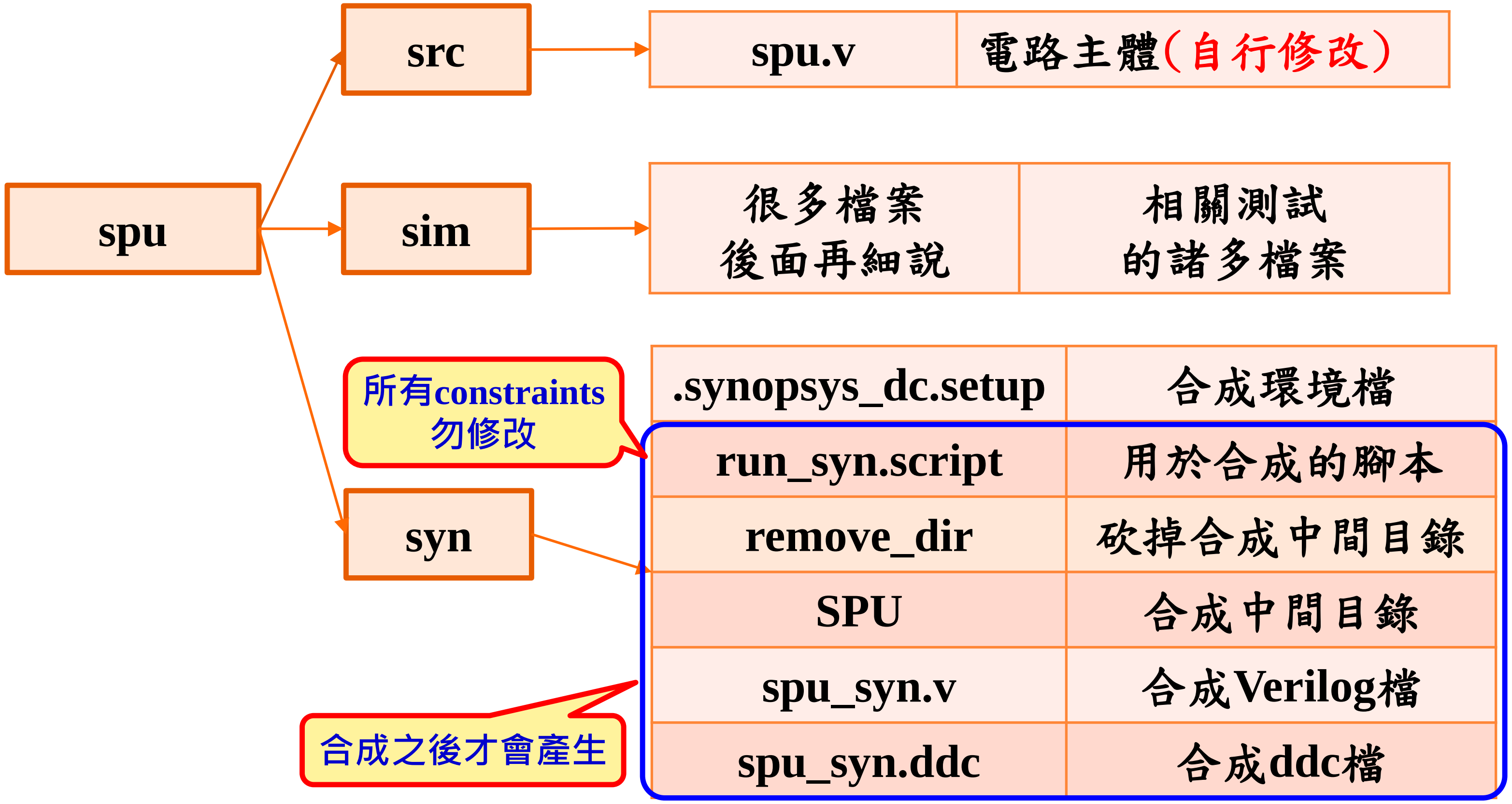
Interface of SPU (Table list)(2/2)

Signal name	I/O	Width	Simple Description
dm_addr	output	8	8-bit data address of data memory
dm_r_data	input	16	16-bit read data of data memory
dm_rd	output	1	read enable of data memory
dm_w_data	output	16	16-bit write data of data memory
dm_wr	output	1	write enable of data memory

FINAL PROJECT：進行方式

登入工作站複製如下目錄：

```
DICS01> cd □ ~/SIP
DICS01> cp □ -r □ /home/C_Share_Linux/spu □ .
```



spu.v	電路主體(自行修改)

很多檔案 後面再細說	相關測試 的諸多檔案

.synopsys_dc.setup	合成環境檔
run_syn.script	用於合成的腳本
remove_dir	砍掉合成中間目錄
SPU	合成中間目錄
spu_syn.v	合成Verilog檔
spu_syn.ddc	合成ddc檔

FINAL PROJECT : SIM 下的檔案

im_data.txt	Instruction memory test data
dm_data.txt	Data memory test data
spu_test.v	合成前的測試檔 (自行修改)
run_sim	合成前的執行測試 shell 檔 DICS01> sh run_sim
spu_syn_test.v	合成後的測試檔 (搭配合成前的測試檔)
run_syn_sim	合成後的執行測試 shell 檔 DICS01> sh run_syn_sim
run_syn_test_fsdb	合成後的執行測試 shell 檔，產出 FSDB 波形檔 DICS01> sh run_syn_sim_fsdb
spu_syn.sdf	合成之後的時序資訊(合成之後產生)
spu_test.fsdb	FSDB 測試波形檔，執行測試 shell 檔之後產生

FINAL PROJECT :繳交資料

- 期末報告檔(PPT)
  - 說明此作業的工作站工作目錄
    - 工作帳號
    - 工作目錄請設定如下: **../../帳號/SIP/spu/ (請按照此命名)**
  - 設計說明(含架構)
  - 電路合成圖(截圖)
  - 電路的 timing (setup & hold)
  - area 的資訊(截圖) : 越小分數越高(只看 cell 的部分)
  - 測試檔說明(須撰寫一段機器碼，測試到所有指令，自由發揮)
  - 合成前與合成後 timing 波形圖(截圖) (截一筆運算即可)
- 請以組長帳號將簡報檔上傳到網路學員 Final Project
  - 其他檔案不用上傳，將所有檔案放在工作站指定目錄下即可
  - Deadline 2021/1/155 23:59
  - **2020/1/14 當天上台報告 (10分鐘) (小組互評)**

FINAL PROJECT：分數價目表

- 老師評分(**40%**)
  - 完成 5-instruction (75分)
  - 完成 6-instruction (80分)
  - 完成 7-instruction (85分)
  - 完成 8-instruction (90分以上，依面積，最高 95分)
- 上台報告(各組互評) (**40%**)
  - 80分~95分
- 組內互評(**10%**)
- 人數(**10%**)
  - 3人: 80分
  - 2人: 85分
  - 1人: 100分

FINAL PROJECT :更新 SIM 與 SYN

- 如果已經有修改這兩目錄下的資料, 先備份自己的資料
- DICS01> cd □ ~/SIP/spu
- DICS01> rm □ –r □ sim
- DICS01> rm □ –r □ syn
- DICS01> cp □ -r □ **/home/C_Share_Linux/spu/sim** □ .
- DICS01> cp □ -r □ **/home/C_Share_Linux/spu/syn** □ .