SIP 111-1 Final Project : A 11-instruction Programmable Processor

Four-Instruction Programmable Processor (Basic)

- Instruction Set – List of allowable instructions and their representation in memory, e.g.,
  - *Load* instruction—0000 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$
    - MOV Ra, d — specifies the operation RF[a]=D[d]
  - *Store* instruction—0001 $r_3r_2r_1r_0$ $d_7d_6d_5d_4d_3d_2d_1d_0$
    - MOV d, Ra — specifies the operation D[d] = RF[a]
  - *Add* instruction—0010 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$
    - ADD Ra, Rb, Rc — specifies the operation RF[a] = RF[b] + RF[c]
  - *Stop* instruction—1111 0000 0000 0000
    - Stop —specifies the operation : stop the processor and reset

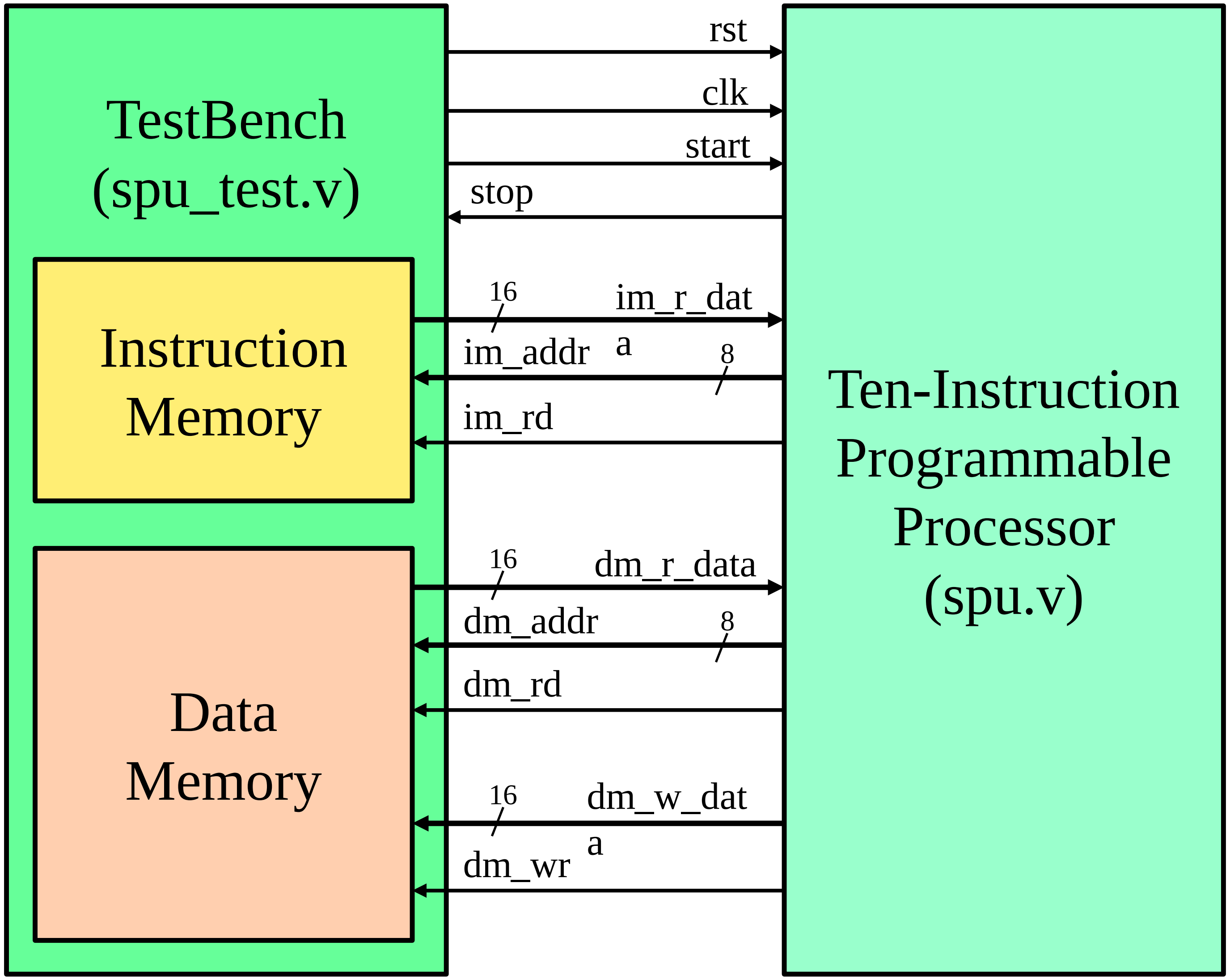
Add 7 more Instructions (1/2)

- Let's add 7 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*
    - For simple implementation, we define that "*c*" is an 8 bits unsigned number.
  - *Add-Immediate* instruction—0100 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $c_3c_2c_1c_0$
    - ADD Ra, Rb, #c—specifies the operation *RF[a]= RF[b] + c*
    - we define that "*c*" is a 4 bits signed number. Note that we need to extend the sign bit before addition if necessary.
  - *Subtract* instruction—0101 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $rc_3rc_2rc_1rc_0$
    - SUB Ra, Rb, Rc—specifies the operation *RF[a]=RF[b] – RF[c]*

ADD 7 MORE INSTRUCTIONS (2/2)

- Let's add 7 more instructions:
  - *Jump-if-zero* instruction—0110 $ra_3ra_2ra_1ra_0$ $o_7o_6o_5o_4o_3o_2o_1o_0$
    - JMPZ Ra, offset—specifies the operation *PC = PC + offset* if *RF[a]* is 0
    - Note that "*offset*" is an 8 bits signed number
  - *Jump-if-not-zero* instruction—0111 $ra_3ra_2ra_1ra_0$ $o_7o_6o_5o_4o_3o_2o_1o_0$
    - JMPNZ Ra, offset—specifies the operation *PC = PC + offset* if *RF[a]* is not 0
    - Note that "*offset*" is an 8 bits signed number
  - *Jump* instruction—1000 0000 $o_7o_6o_5o_4o_3o_2o_1o_0$
    - JMP offset —specifies the operation *PC = PC + offset*
    - Note that "*offset*" is an 8 bits signed number
  - *Jump-if-equal* instruction—1001 $ra_3ra_2ra_1ra_0$ $rb_3rb_2rb_1rb_0$ $o_3o_2o_1o_0$
    - JMPEQ Ra, Rb, offset
      —specifies the operation *PC = PC + offset* if *RF[a] == RF[b]*
    - Note that "*offset*" is an 4 bits signed number (2's complement)
      Note that we need to extend the sign bit before addition if necessary.

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

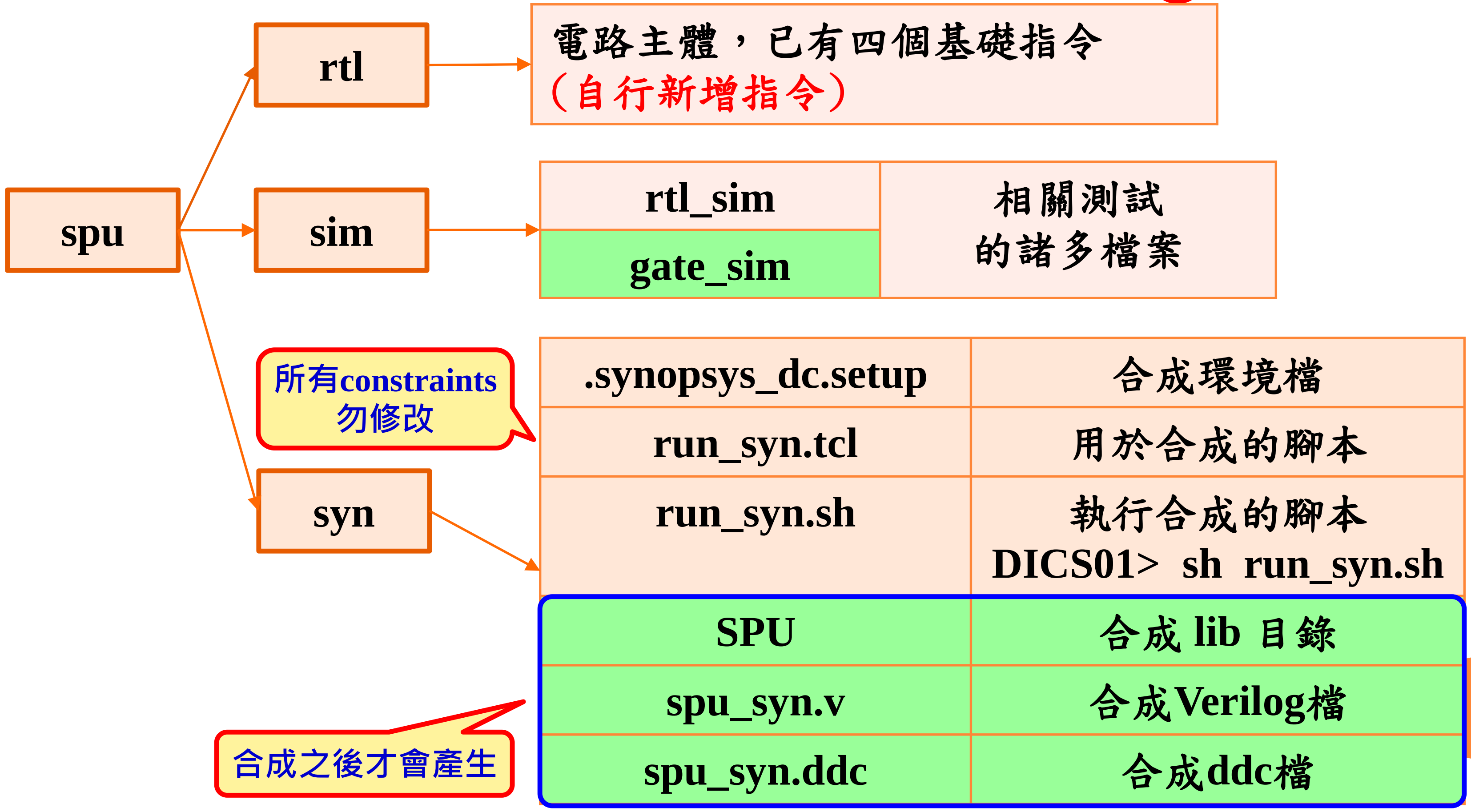
Interface of SPU (Table list)(2/2)

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 □ .
```



rtl	
電路主體，已有四個基礎指令（自行新增指令）	

sim	
rtl_sim	相關測試的諸多檔案
gate_sim	

syn	
.synopsys_dc.setup	合成環境檔
run_syn.tcl	用於合成的腳本
run_syn.sh	執行合成的腳本 DICS01> sh run_syn.sh
SPU	合成 lib 目錄
spu_syn.v	合成Verilog檔
spu_syn.ddc	合成ddc檔

FINAL PROJECT：RTL_SIM 下的檔案

tb/	spu_tb.v	合成前的測試檔 (自行修改)
run/	im_data.txt	instruction memory test data (自行修改，合成前與合成後可以用相同檔案)
	dm_data.txt	Data memory test data (自行修改，合成前與合成後可以用相同檔案)
	dm_ex_out.txt	測試後(合成前與合成後)，Data memory updated data
	run.f	模擬使用的 run.f
	run_sim.sh	合成前的執行測試 shell 檔 DICS01> sh run_sim.sh run.f
	spu_tb.fsdb	合成前的FSDB 測試波形檔，執行 “run_sim.sh” shell 檔之後產生

FINAL PROJECT：GATE_SIM 下的檔案

tb/	spu_syn_tb.v	合成後的測試檔 (自行修改，搭配合成前的測試檔)
run/	im_data.txt	instruction memory test data (自行修改，合成前與合成後可以用相同檔案)
	dm_data.txt	Data memory test data (自行修改，合成前與合成後可以用相同檔案)
	dm_ex_out.txt	測試後(合成前與合成後)，Data memory updated data
	run_syn_sim.sh	合成後的執行測試 shell 檔 DICS01> sh run_syn_sim
	spu_syn_tb.fsdb	合成後的FSDB 測試波形檔，執行 “run_syn_sim.sh” shell 檔之後產生
	spu_syn.sdf	合成後的時序資訊

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(一個組員帳號為代表)
    - 工作目錄請設定如下: **../../帳號/SIP/spu/ (請按照此命名)**
    - 目錄說明
  - 設計說明(含架構)(詳述完成哪些指令)
  - 電路合成圖(截圖)
  - 電路的 timing (setup & hold)
  - area 的資訊(截圖)
  - 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)
  - **各組員工作項目與貢獻度**
- 簡報檔上傳到 Eclass 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 2022/1/12 00:10
  - **2022/1/12 當天上台報告 (5~8分鐘) (小組互評)**

FINAL PROJECT：分數標準

- 實作評分(**60%**)
  - 無法完成任何新增指令(4-instruction(70分)**(但請改變測試程式碼)**
  - 完成 5-instruction (75分)
  - 完成 6-instruction (80分)
  - 完成 7-instruction (85分)
  - 完成 8-instruction (90分)
  - 完成 9-instruction (92分)
  - 完成 10-instruction (94分)
  - 完成 11-instruction (96分)
- 上台報告(各組互評) (**30%**)
  - 80分~95分
- 小組人數(**10%**)
  - 3人 (70分)
  - 2人 (80分)
  - 1人 (100分)

上台報告(各組互評)

項目	普通 (80)	不錯(85)	好(90)	讚(95)
口條表達 (40%)	沒甚麼特殊之處	口條清晰	口條清晰, 表達清楚	口條清晰, 表達清楚, 台風穩健.
報告內容 (60%) (不考慮實現的指令數,只針對是否有將應呈現的部分完整表示出來)	沒甚麼特殊之處	內容完整	內容完整,測試機械碼能驗證功能, 簡報圖文賞心悅目	內容完整,測試機械碼能驗證功能,且考慮周延. 簡報圖文賞心悅目有巧思