Explanation for Homework IV

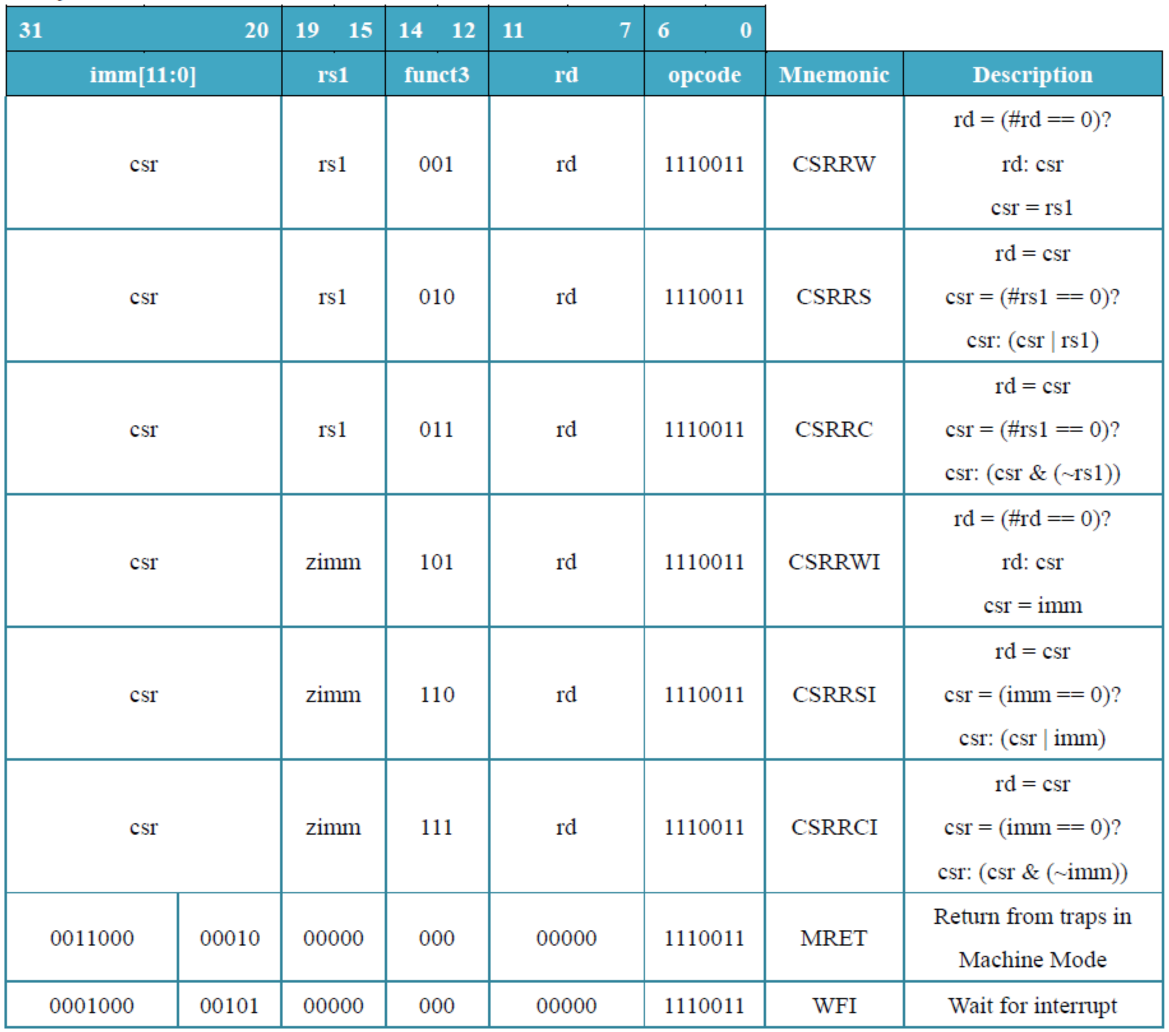
Advisor: Lih-Yih Chiou
Speaker: John
Date: 2018/11/29

Outline

- New Instructions
- CSR
- Slaves
- Simulation
- Verification

New Instructions

System



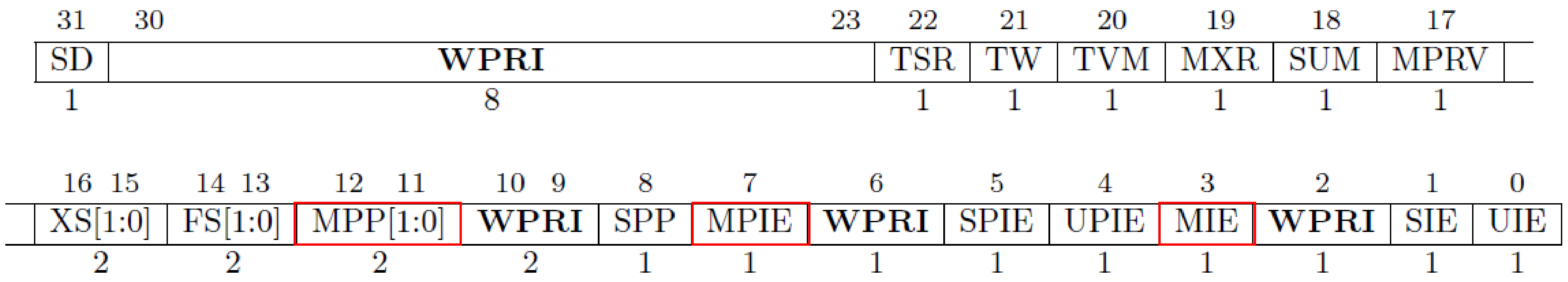
31	20	19 15	14 12	11 7	6 0		
imm[11:0]		**rs1**	**funct3**	**rd**	**opcode**	**Mnemonic**	**Description**
csr		rs1	001	rd	1110011	CSRRW	rd = (#rd == 0)? rd: csr csr = rs1
csr		rs1	010	rd	1110011	CSRRS	rd = csr csr = (#rs1 == 0)? csr: (csr | rs1)
csr		rs1	011	rd	1110011	CSRRC	rd = csr csr = (#rs1 == 0)? csr: (csr & (~rs1))
csr		zimm	101	rd	1110011	CSRRWI	rd = (#rd == 0)? rd: csr csr = imm
csr		zimm	110	rd	1110011	CSRRSI	rd = csr csr = (imm == 0)? csr: (csr | imm)
csr		zimm	111	rd	1110011	CSRRCI	rd = csr csr = (imm == 0)? csr: (csr & (~imm))
0011000	00010	00000	000	00000	1110011	MRET	Return from traps in Machine Mode
0001000	00101	00000	000	00000	1110011	WFI	Wait for interrupt

CSR

Table 1-4: Control Status Register (CSR)

Address	Privilege	Name	Requirement
0x300	MRW	mstatus	MIE, MPIE, MPP. Others hardwire to 0
0x304	MRW	mie	MEIE. Others hardwire to 0
0x305	MRW	mtvec	Hardwire to 32'h0001_0000
0x341	MRW	mepc	MEPC[31:2]. Others hardwire to 0
0x344	MRW	mip	MEIP. Others hardwire to 0
0xB00	MRW	mcycle	All
0xB02	MRW	minstret	All
0xB80	MRW	mcycleh	All
0xB82	MRW	minstreth	All

CSR – mstatus



31	30 … 23	22	21	20	19	18	17
SD	**WPRI**	TSR	TW	TVM	MXR	SUM	MPRV
1	8	1	1	1	1	1	1

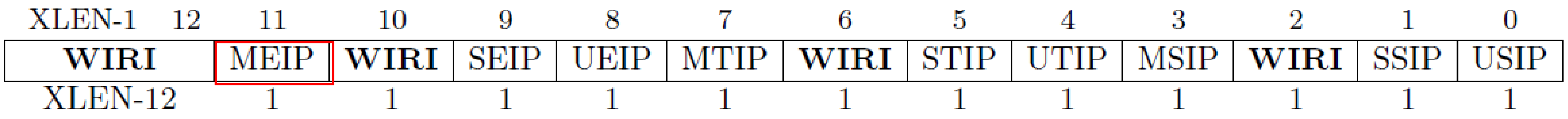
16 15	14 13	12 11	10 9	8	7	6	5	4	3	2	1	0
XS[1:0]	FS[1:0]	MPP[1:0]	**WPRI**	SPP	MPIE	**WPRI**	SPIE	UPIE	MIE	**WPRI**	SIE	UIE
2	2	2	2	1	1	1	1	1	1	1	1	1

Figure 3.6: Machine-mode status register (`mstatus`) for RV32.

- When interrupt is taken
  - Priv ← 2'b11 (Machine Mode)
  - MPIE ← MIE
  - MIE ← 0
  - MPP ← 2'b11 (Machine Mode)
- When interrupt return
  - Priv ← MPP
  - MPIE ← 1
  - MIE ← MPIE
  - MPP ← 2'b11 (Machine Mode)

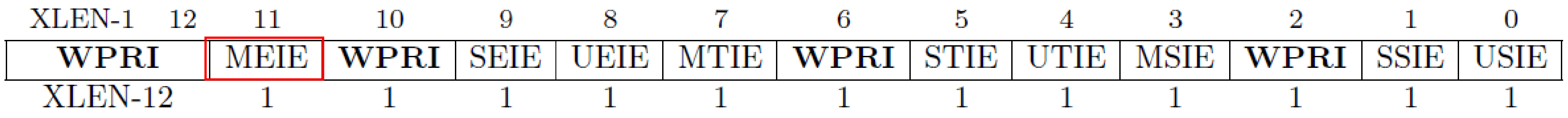
★ WFI instruction should ignore MIE

CSR – mie & mip



XLEN-1 12	11	10	9	8	7	6	5	4	3	2	1	0
WIRI	MEIP	**WIRI**	SEIP	UEIP	MTIP	**WIRI**	STIP	UTIP	MSIP	**WIRI**	SSIP	USIP
XLEN-12	1	1	1	1	1	1	1	1	1	1	1	1

Figure 3.11: Machine interrupt-pending register (`mip`).



XLEN-1 12	11	10	9	8	7	6	5	4	3	2	1	0
WPRI	MEIE	**WPRI**	SEIE	UEIE	MTIE	**WPRI**	STIE	UTIE	MSIE	**WPRI**	SSIE	USIE
XLEN-12	1	1	1	1	1	1	1	1	1	1	1	1

Figure 3.12: Machine interrupt-enable register (`mie`).

- Interrupt is pending
  - MEIP = 1
- Enable local interrupt
  - MEIE = 1

★ WFI instruction shouldn't ignore MEIE

CSR – mtvec & mepc

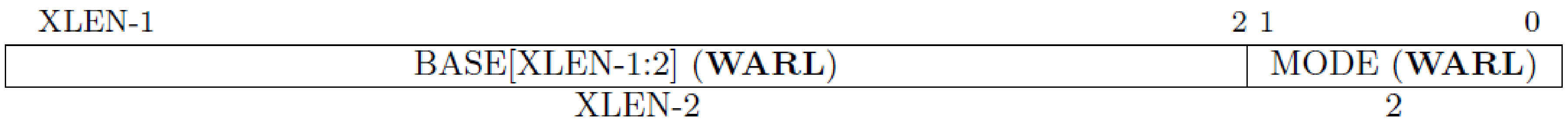


Figure 3.8: Machine trap-vector base-address register (`mtvec`).

- Fix trap-vector to 0x0001_0000 (IM start address)

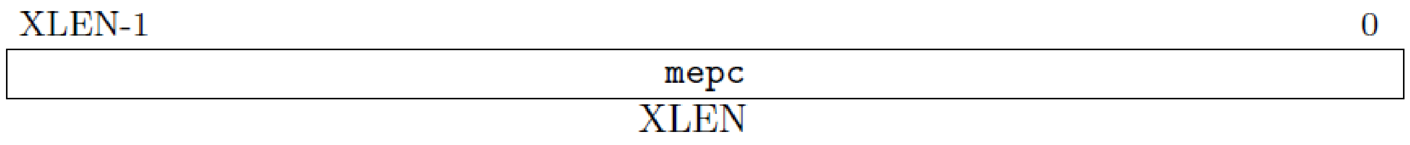
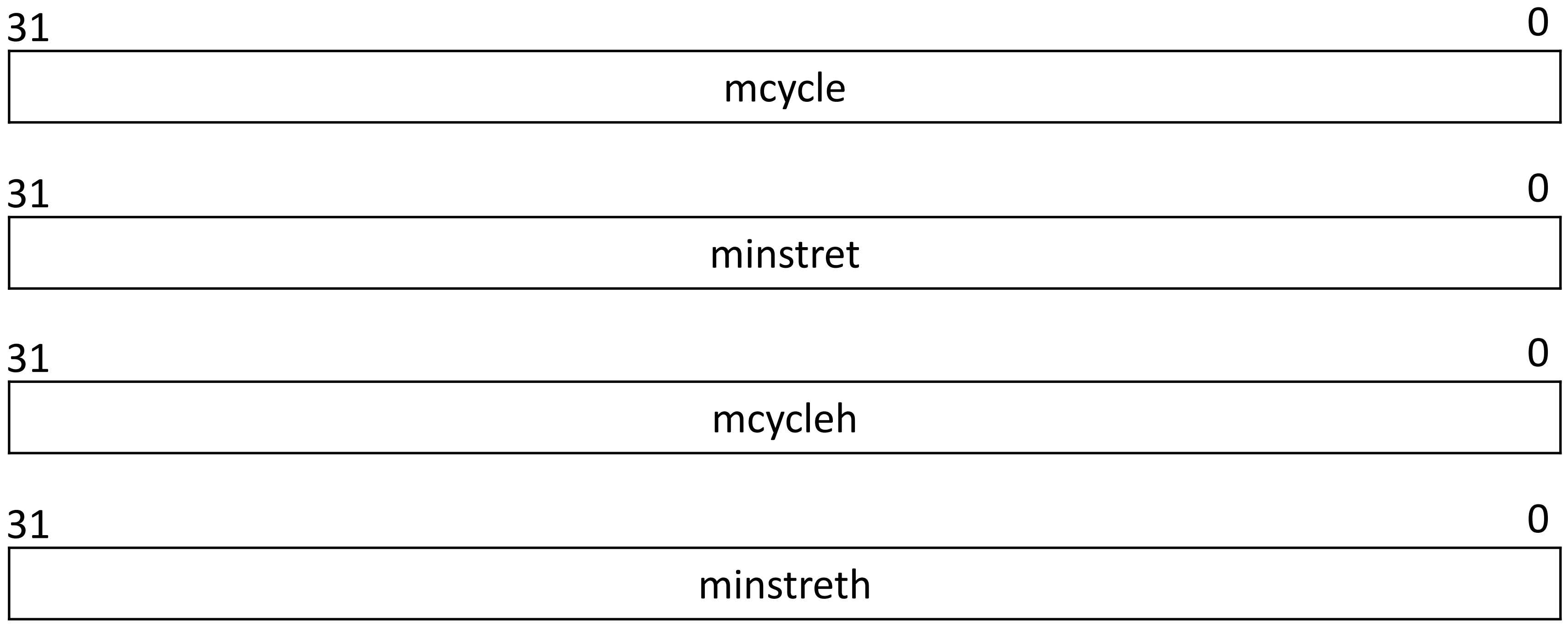


Figure 3.20: Machine exception program counter register.

- When interrupt is taken
  - MEPC ← PC + ? (Decided by implementation)
  - PC ← {MTVEC[31:2], 2’b00}
- When interrupt return
  - PC ← MEPC

CSR - Hardware Performance Monitor



- Actually two 64-bit counter **cycle** & **instret**
  - Cycle increase every cycle
  - Instret increase when instruction retire

Slave Configuration

Table 1-5 : Slave configuration

NAME	**Number**	**Start address**	**End address**
ROM	Slave 0	0x0000_0000	0x0000_1FFF
IM	Slave 1	0x0001_0000	0x0001_FFFF
DM	Slave 2	0x0002_0000	0x0002_FFFF
sensor_ctrl	Slave 3	0x1000_0000	0x1000_03FF
DRAM	Slave 4	0x2000_0000	0x201F_FFFF

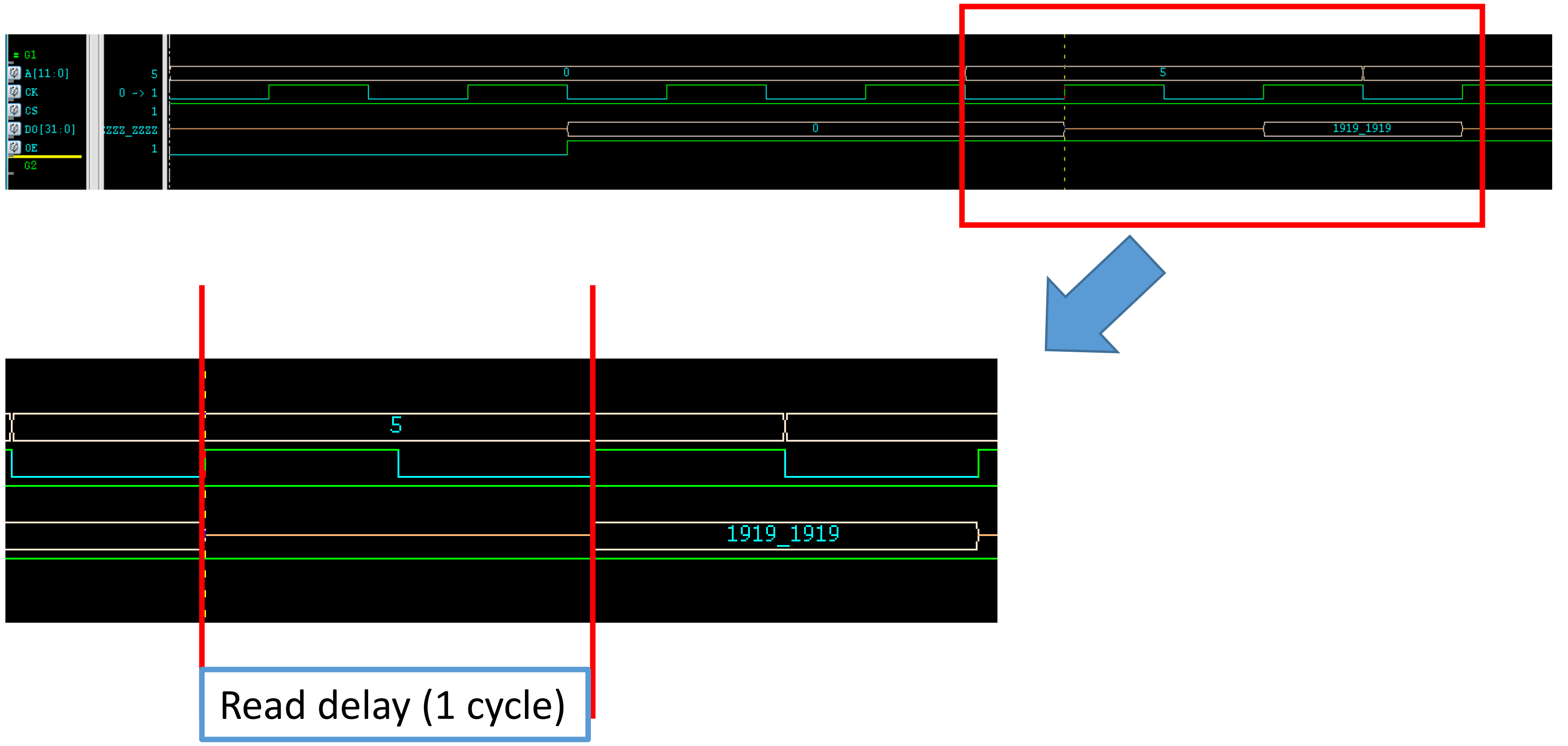
- You should design all slave wrappers !!

ROM (1)

ROM	System signals			
	CK	input	1	System clock
	Memory ports			
	DO	output	32	ROM data output
	OE	input	1	Output enable (active high)
	CS	input	1	Chip select (active high)
	A	input	12	ROM address input
	Memory Space			
	Memory_byte0	reg	8	Size: [0:1023]
	Memory_byte1	reg	8	Size: [0:1023]
	Memory_byte2	reg	8	Size: [0:1023]
	Memory_byte3	reg	8	Size: [0:1023]

ROM (2)

- ROM_tb.sv
  - Read



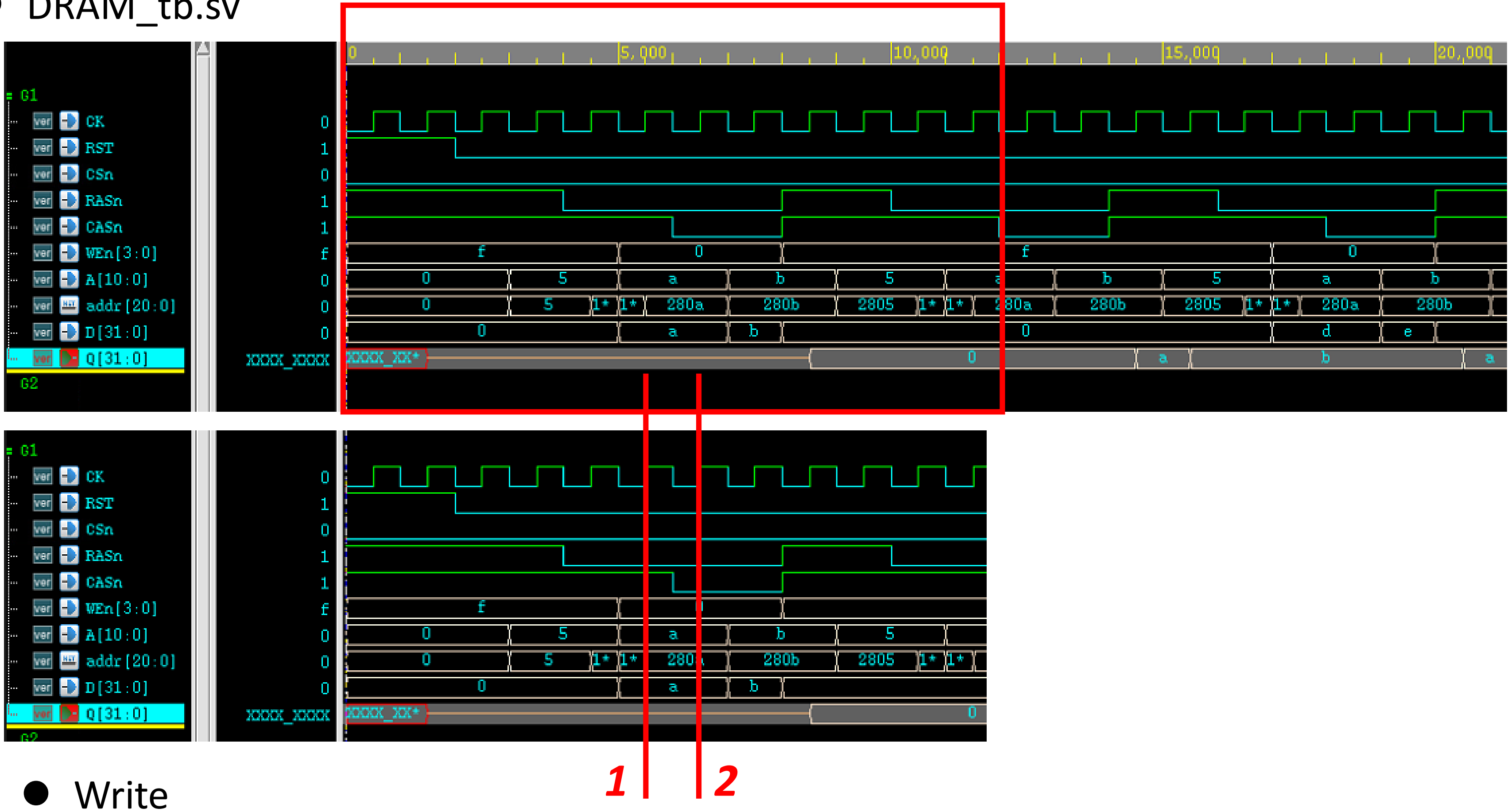
DRAM (1)

DRAM	System signals			
	CK	input	1	System clock
	RST	input	1	System reset (active high)
	Memory ports			
	CSn	input	1	DRAM Chip Select (active low)
	WEn	input	4	DRAM Write Enable (active low)
	RASn	input	1	DRAM Row Access Strobe (active low)
	CASn	input	1	DRAM Column Access Strobe (active low)
	A	input	11	DRAM Address input
	D	input	32	DRAM data input
	Q	output	32	DRAM data output
	Memory space			
	Memory_byte0	reg	8	Size: [0:2097151]
	Memory_byte1	reg	8	Size: [0:2097151]
	Memory_byte2	reg	8	Size: [0:2097151]
	Memory_byte3	reg	8	Size: [0:2097151]

★ Row address is 11-bit and column address is 10-bit

DRAM (2)

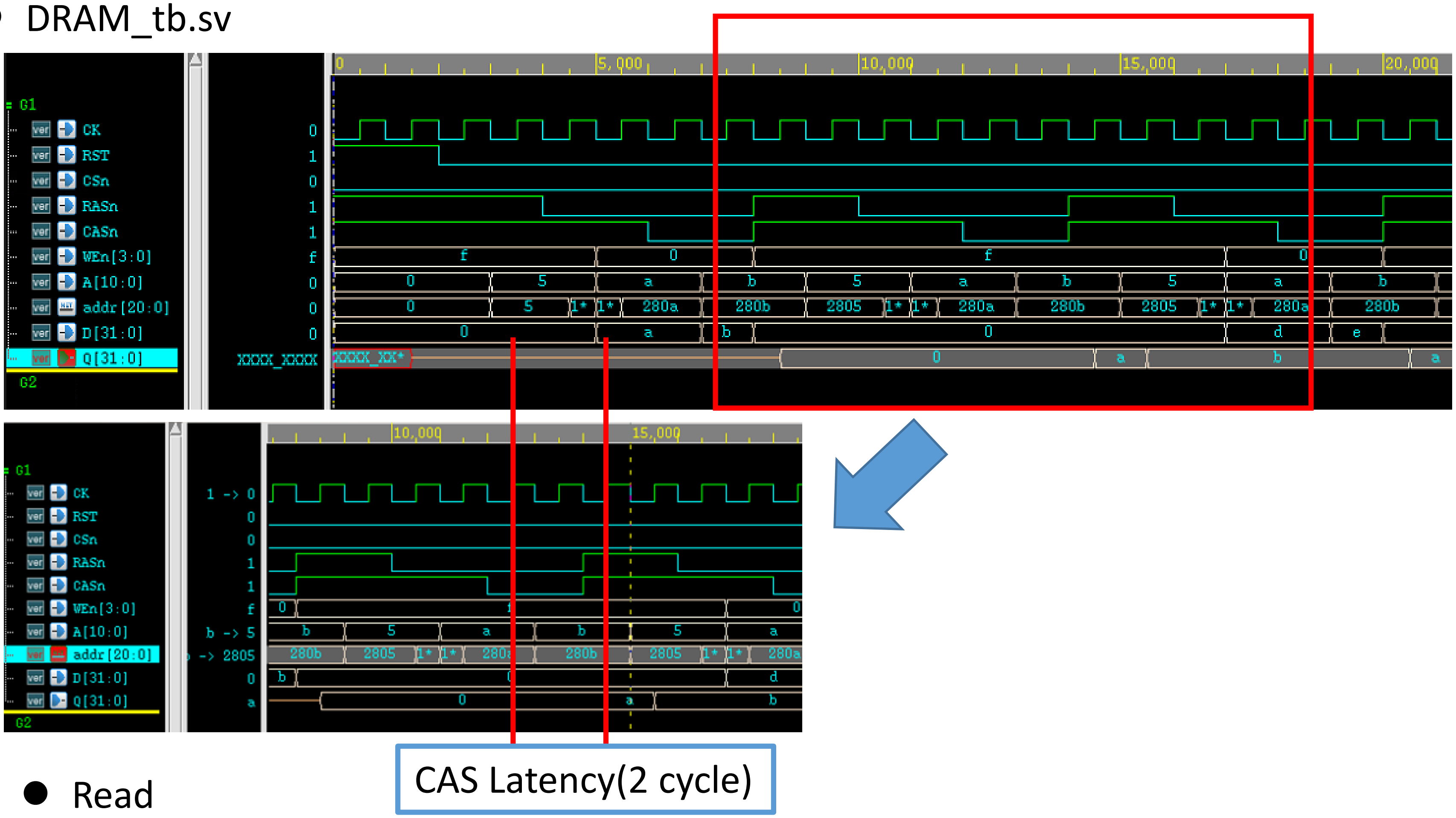
- DRAM_tb.sv



- Write
  - RAS → Set Row (A should be ready) *1*
  - CAS → Set Column (A, WEn and D should be ready) *2*

DRAM (3)

- DRAM_tb.sv



- Read
  - RAS → Set Row (A should be ready)
  - CAS → Set Column (A and WEn should be ready)

Sensor Control (1)

sensor_ctrl	System signals			
	clk	input	1	System clock
	rst	input	1	System reset (active high)
	sctrl_en	input	1	Sensor controller enable (active high)
	sctrl_clear	input	1	Sensor controller clear (active high)
	sctrl_addr	input	6	Sensor controller address
	sctrl_interrupt	output	1	Sensor controller interrupt
	sctrl_out	output	32	Sensor controller data output
	sensor_ready	input	1	Sensor data ready
	sensor_out	input	32	Data from sensor
	sensor_en	output	1	Sensor enable (active high)
	Memory space			
	mem	logic	32	Size: [0:63]

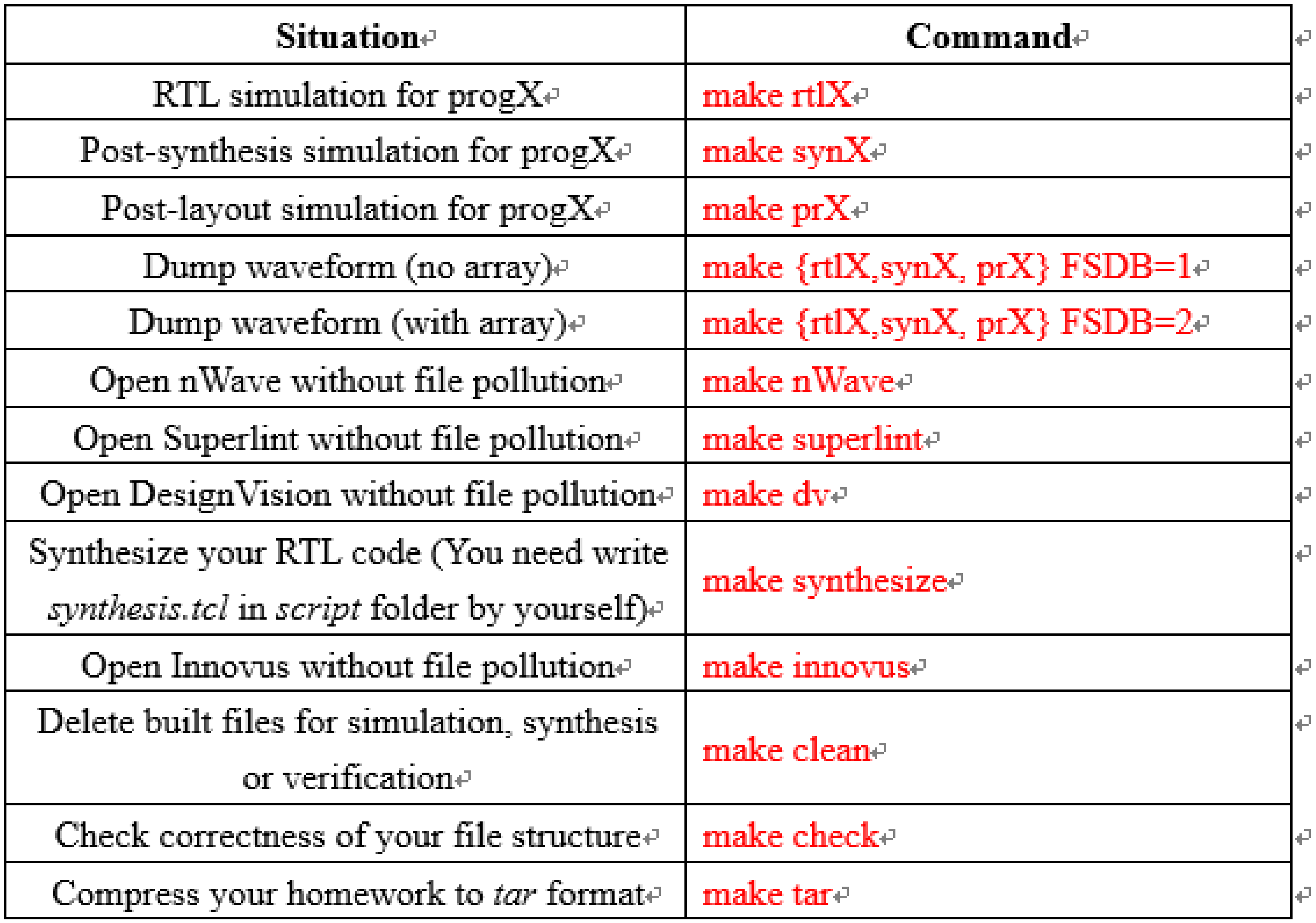
Sensor Controller (2)

- Sensor generates a new data every 1024 cycles
- Sensor controller stores data to its local memory
- When local memory is full (64 data), sensor controller will stop requesting data (sensor_en = 0) and assert interrupt (sctrl_interrupt = 1)
- CPU load data from sensor controller and store it to DM
- Write non-zero value in 0x1000_0100 or 0x1000_0200 to enable stcrl_en or stcrl_clear

Address	Mapping
0x1000_0000 – 0x1000_00FF	mem[0] – mem[63]
0x1000_0100	stcrl_en
0x1000_0200	stcrl_clear

Simulation

Table B-2: Makefile macros

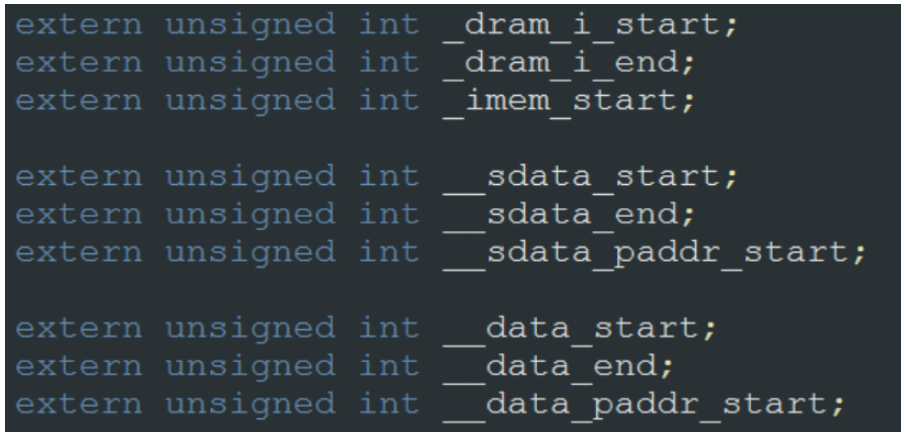


Situation	Command
RTL simulation for progX	make rtlX
Post-synthesis simulation for progX	make synX
Post-layout simulation for progX	make prX
Dump waveform (no array)	make {rtlX,synX, prX} FSDB=1
Dump waveform (with array)	make {rtlX,synX, prX} FSDB=2
Open nWave without file pollution	make nWave
Open Superlint without file pollution	make superlint
Open DesignVision without file pollution	make dv
Synthesize your RTL code (You need write *synthesis.tcl* in *script* folder by yourself)	make synthesize
Open Innovus without file pollution	make innovus
Delete built files for simulation, synthesis or verification	make clean
Check correctness of your file structure	make check
Compress your homework to *tar* format	make tar

Verification (1/6)

- Prog0: booting program & given instruction verification
  - Copy ISR and main program to destination and jump to main program. This is the only code you should write and store in ROM. Your program counter should start at 0x0000_0000
  - Start at PC=0x0000_0000 [setup.S is already provided by TA]
  - **Write your boot.c/boot.s by yourself to move data from DRAM to IM + DM**
  - Run main.S [main.S is already provided by TA)].

> - _dram_i_start = instruction start address in DRAM.
> - _dram_i_end = instruction end address in DRAM.
> - _imem_start = instruction start address in IM

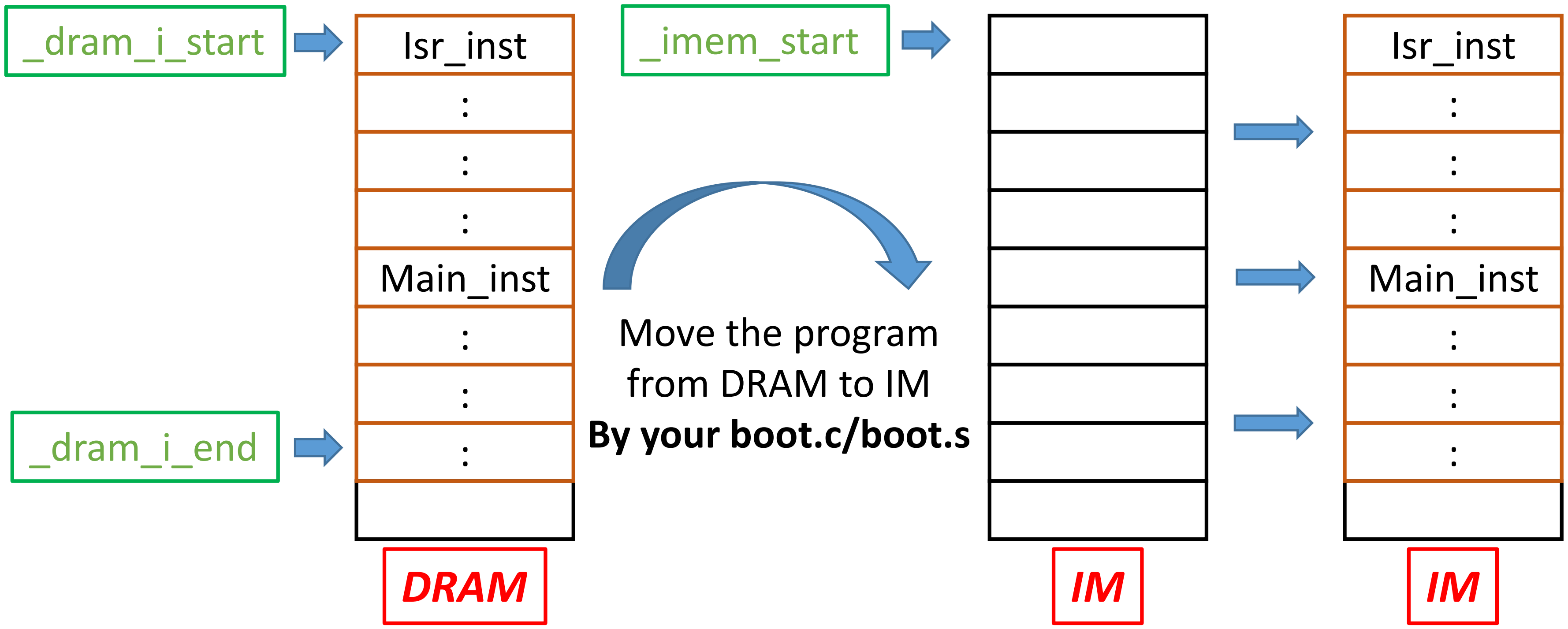


```
extern unsigned int _dram_i_start;
extern unsigned int _dram_i_end;
extern unsigned int _imem_start;

extern unsigned int __sdata_start;
extern unsigned int __sdata_end;
extern unsigned int __sdata_paddr_start;

extern unsigned int __data_start;
extern unsigned int __data_end;
extern unsigned int __data_paddr_start;
```

Verification (2/6)

Prog0: booting program & given instruction verification

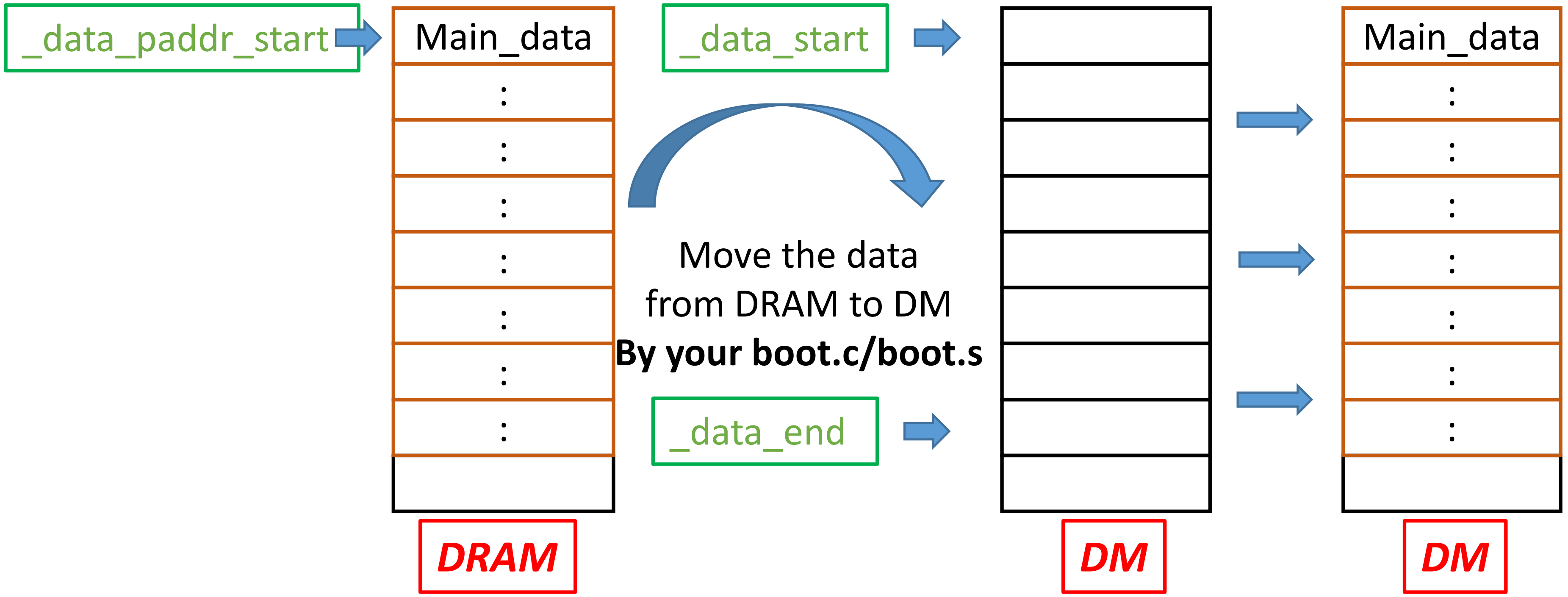
- _dram_i_start = instruction start address in DRAM.
- _dram_i_end = instruction end address in DRAM.
- _imem_start = instruction start address in IM



Verification (3/6)

- Prog0: booting program & given instruction verification

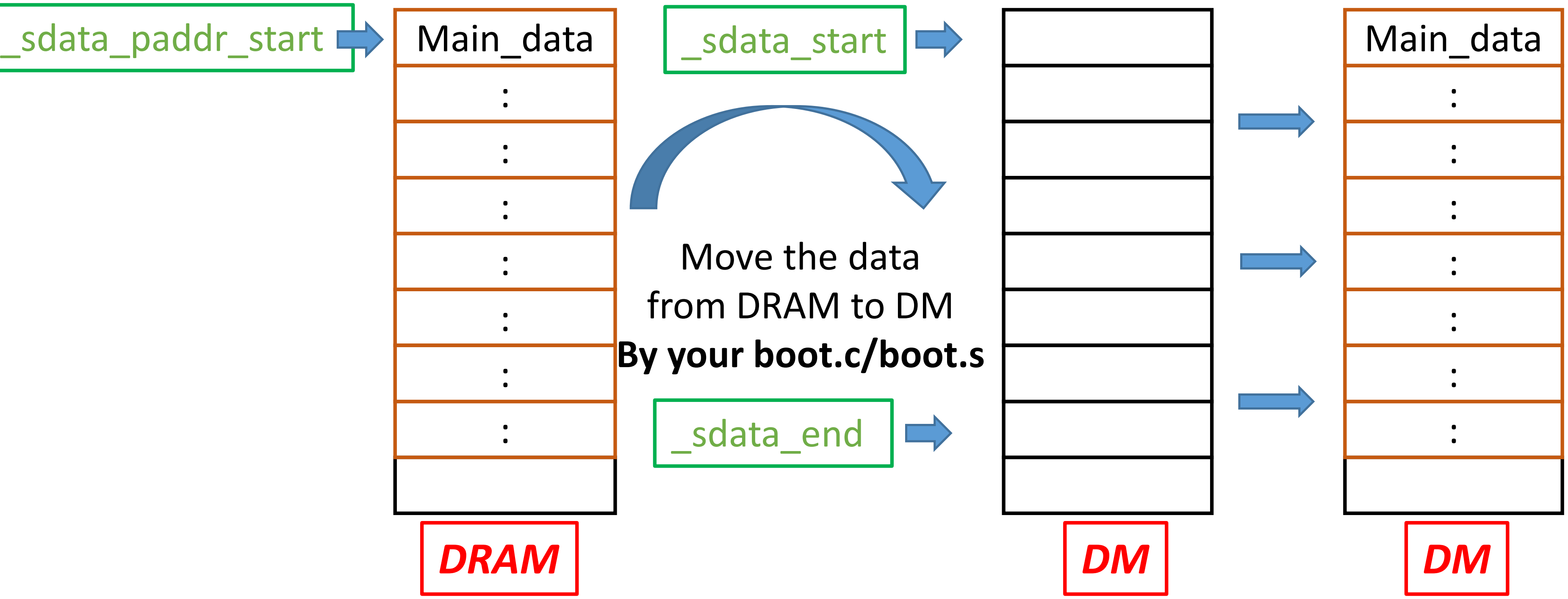
> ➢ _data_start = Main_data start address in DM.
> ➢ _data_end = Main_data end address in DM.
> ➢ _data_paddr_start = Main_data start address in DRAM



Verification (4/6)

Prog0: booting program & given instruction verification

- _sdata_start = Main_data start address in DM.
- _sdata_end = Main_data end address in DM.
- _sdata_paddr_start = Main_data start address in DRAM



Verification (5/6)

- Prog1: booting program & Interrupt verification
  - Boot like prog0
  - Main program will setup environment and activate sensor controller
  - When sensor controller interrupt CPU, ISR will be activated and copy data to DM. Then, reset the counter of sensor controller
  - When copy is done, ISR jump back to main program and sort these data
  - After 4 groups of data are sorted, the simulation will complete.

Verification (6/6)

Prog1 (Process)

- Start at PC=0x0000_0000 [setup.S is already provided by TA]
- Write your boot.c/boot.s by yourself to move data from DRAM to IM + DM
- Run main.S [main.S is already provided by TA)]
- Execution order:
  1) main.c (main/sort function)
  2) isr.S (provided by TA)
  3) main.c (copy function)
  4) isr.S
  5) main.c (sort function)