《计算机组成》课程实验之单周期 CPU 模块设计仿真
要求说明
202004

本课程实验教学目的，是要求同学们利用 Verilog 硬件描述语言、基于大规模可编程逻辑阵列器件 FPGA，进行计算机 CPU 的逻辑功能设计，包括 RISC 风格的单周期 CPU 设计、5 级流水 CPU 设计、以及在其上的输入输出部件扩展和功能扩展。

以下是第一部分单周期 CPU 模块设计的实验目的与要求、提供给同学的设计文件说明、仿真要求以及设计指导。

一、实验目的和要求

1、采用 Verilog 硬件描述语言在 Quartus II EDA 设计平台中，基于 Intel cyclone II 系列 FPGA 完成具有执行 20 条 MIPS 基本指令的单周期 CPU 模块的设计。根据提供的单周期 CPU 示例程序的 Verilg 代码文件，将设计代码补充完整，实现该模块的电路设计。

2、利用实验提供的标准测试程序代码，完成单周期 CPU 模块的功能仿真测试，验证 CPU 执行所设计的 20 条 RISC 指令功能的正确性。

从而理解计算机五大组成部分的协调工作原理，理解存储程序自动执行的原理和掌握运算器、存储器、控制器的设计和实现原理。

二、文件说明

1、提供的单周期 CPU 模块示例程序源代码包含以下具体文件，如图 1 。

alu.v	2011/5/10 21:33	V 文件	2 KB
cla32.v	2020/4/15 18:32	V 文件	1 KB
dff32.v	2014/4/2 16:50	V 文件	1 KB
dffe32.v	2009/9/14 15:53	V 文件	1 KB
lpm_ram_dq_dram	2011/5/10 13:39	BSF 文件	3 KB
lpm_ram_dq_dram.v	2011/5/10 13:39	V 文件	8 KB
lpm_rom_irom	2011/5/10 13:22	BSF 文件	3 KB
lpm_rom_irom.v	2011/5/10 13:22	V 文件	7 KB
mux2x5.v	2009/9/17 14:49	V 文件	1 KB
mux2x32.v	2009/9/14 14:09	V 文件	1 KB
mux4x32.v	2009/9/15 15:11	V 文件	1 KB
Real_Value_Table_to student	2011/5/4 9:35	Microsoft Excel ...	28 KB
regfile.v	2011/5/11 2:01	V 文件	1 KB
sc_computer.v	2011/5/11 11:31	V 文件	1 KB
sc_computer_test_wave_01.vwf	2020/4/15 17:56	VWF 文件	156 KB
sc_cpu.v	2020/4/15 18:28	V 文件	2 KB
sc_cu.v	2014/3/31 13:51	V 文件	2 KB
sc_datamem.mif	2009/9/17 20:13	MIF 文件	1 KB
sc_datamem.v	2011/5/10 22:55	V 文件	1 KB
sc_instmem.mif	2009/9/14 16:06	MIF 文件	3 KB
sc_instmen.v	2011/5/11 11:28	V 文件	1 KB

图 1 提供的单周期 CPU 模块示例程序源代码

其中 **sc_computer** 为顶层文件。自行定义**同名**工程文件，将以上所有设计所需源文件文件 copy 到工作目录并添加入工程。如图 2（a）。顶层文件代码如下：

```
module sc_computer (resetn,clock,mem_clk,pc,inst,aluout,memout,imem_clk,dmem_clk);
   //定义顶层模块  sc_computer，作为工程文件入口
```

```
input resetn,clock,mem_clk;
 //定义整个计算机 module 和外界交互的输入信号，包括复位信号 resetn，时钟信号
 //clock、以及一个频率是 clock 两倍的 mem_clk 信号。这些信号都可以用作仿真验
 //证时的输出观察信号。
output [31:0] pc,inst,aluout,memout;
 //模块用于仿真输出的观察信号。缺省为 wire 型。
output        imem_clk,dmem_clk;
 //模块用于仿真输出的观察信号，用于观察指令 ROM 和数据 RAM 的读写时序。
wire    [31:0] data; //模块间互联传递数据活控制信息的信号线。
wire           wmem;   //模块间互联传递数据活控制信息的信号线。all these
"wire"s are used to connect or interface the cpu,dmem,imem and so on.
sc_cpu cpu (clock,resetn,inst,memout,pc,wmem,aluout,data);
 // CPU module.
 //实例化了一个 CPU，内部包含运算器 ALU 模块，控制器 CU 模块等。
 //在 CPU 模块的原型定义 sc_cpu 模块中，可看到其内部的各个模块构成。
sc_instmem   imem (pc,inst,clock,mem_clk,imem_clk);
 // instruction memory.
 //指令 ROM 存储器 imem 模块。模块原型由 sc_instmem 定义。
 //由于 Cyclone 系列 FPGA 只能支持同步的 ROM 和 RAM，读取操作需要时钟信
 //号。
 //示例代码中采用 quartus 提供的 ROM 宏模块 lpm_rom 实现的，需要读取时钟，
 //该 imem_clk 读取时钟由 clock 信号和 mem_clk 信号组合而成，具体时序可参考模
 //块内相应代码。imem_clk 作为模块输出信号共仿真器进行观察。
sc_datamem   dmem (aluout,data,memout,wmem,clock,mem_clk,dmem_clk );
 // data memory.
 //数据 RAM 存储器 dmem 模块。模块原型由 sc_datamem 定义。
 //由于 Cyclone 系列 FPGA 只能支持同步的 ROM 和 RAM，读取操作需要时钟信
 //号。
 //示例代码中采用 quartus 提供的 RAM 宏模块 lpm_ra_dq 实现的，需要读写时钟，
 //该 dmem_clk 读写时钟由 clock 信号和 mem_clk 信号组合而成，具体时序可参考
 //模块内相应代码。dmem_clk 同时作为模块输出信号共仿真器进行观察。

Endmodule
```

以上顶层代码表明，该单周期 sc_computer 由 CPU、指令 ROM、数据 RAM 共三个模块组成。三个模块之间由定义为 wire 型连接信号线的多路数据连接信号线通过各个模块的输入输出端口（信号）进行互联。同层同名的信号线连通不同模块的端口，一个模块的输出信号可作为一个或者多个模块的输入信号。编写代码时注意定义时的线宽需保持一致。

本部分实验已经提供了几乎全部的实验代码，设计补齐需要填充的代码后，尽快熟悉和精通 Quartus II 下基于 Verilog HDL 的系统设计方法。需要补充完整如下两个文件里的代码段：

alu.v

sc_cu.v

文件 Real_Value_Table_to student.xls 可协助在设计并补充上面两个代码段时参考使用。不用加进工程里。

该实验的示例代码是学习基础，后续两个实验中，流水线 CPU 及 IO 模块的 verilog 代码，主要由同学自行完成，并进行自主的创新设计。希望由你采用你自己设计的流水线 CPU，

利用你设计的 I/O 控制端口，通过运行程序，展现你的设计。

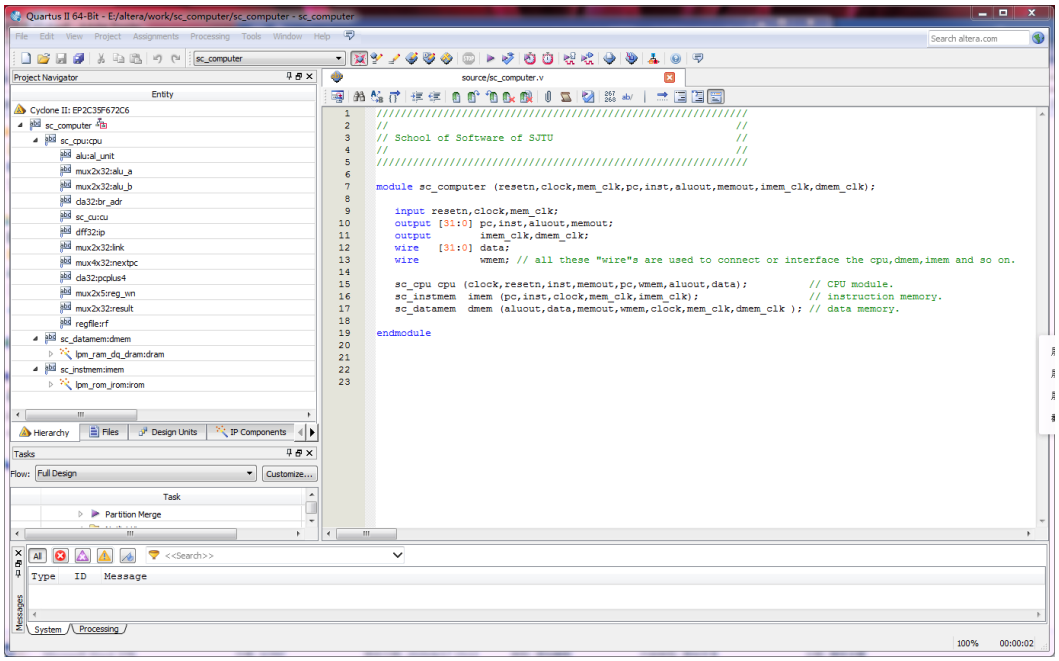

图 2（a）工程文件所包含的相关文件及顶层文件代码示例

2、提供了一个用于功能仿真的激励信号波形文件：sc_computer_test_wave_01.vwf。可用该文件进行仿真验证。仿真调试过程中可根据调试需要基于该文件进行修改。

打开波形激励文件，如图 2（b）。注意 3 个输入信号 resetn、clock、mem_clk 的时序关系。为观察直观，其它的输出及内部观测信号 imem_clk、dmem_clk、pc、inst、aluout、memout、CPU 内寄存器等设置为“uninitialized”状态。

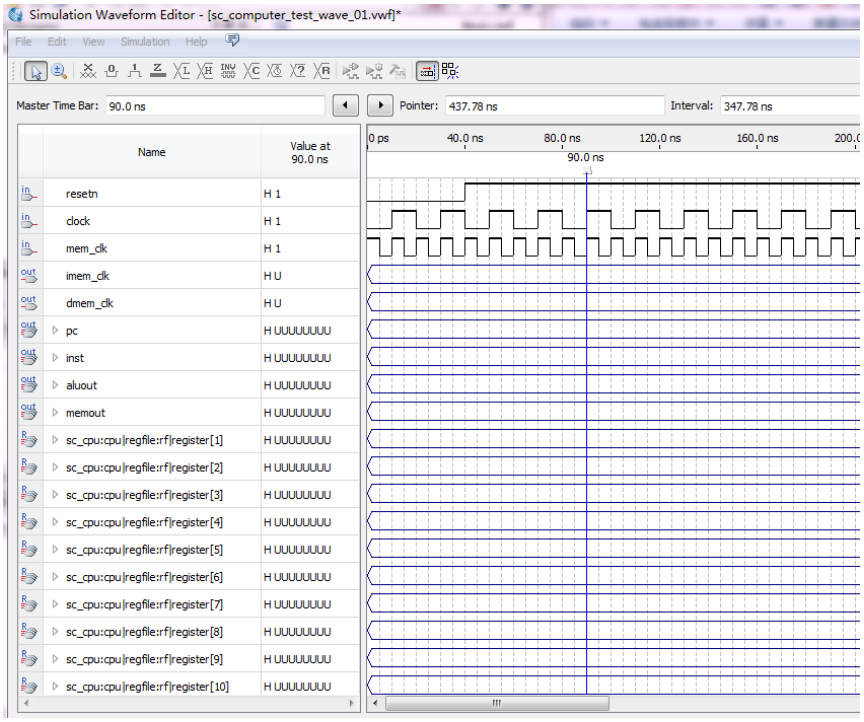

图 2（b） 建立的仿真运行测试输入文件波形

3、sc_instmem.mif 文件为指令存储器初始值文件，该段示例的 CPU 程序代码，涵盖了所有

应实现而待验证的 RISC 指令类型。

文件里每行对应了存储单元字顺序号、机器指令、%（PC 值）、汇编指令、#语句注释%。具体内容如下：

```
DEPTH = 64; % Memory depth and width are required %
WIDTH = 32; % Enter a decimal number %
ADDRESS_RADIX = HEX; % Address and value radixes are optional %
DATA_RADIX = HEX; % Enter BIN, DEC, HEX, or OCT; unless %
% otherwise specified, radixes = HEX %
CONTENT
BEGIN
 0 : 3c010000; % (00) main:     lui $1, 0                    # address of data[0] %
 1 : 34240050; % (04)            ori $4, $1, 80               # address of data[0] %
 2 : 20050004; % (08)            addi $5, $0, 4               # counter %
 3 : 0c000018; % (0c) call:     jal sum                      # call function %
 4 : ac820000; % (10)            sw $2, 0($4)                 # store result %
 5 : 8c890000; % (14)            lw $9, 0($4)                 # check sw %
 6 : 01244022; % (18)            sub $8, $9, $4               # sub: $8 <- $9 - $4 %
 7 : 20050003; % (1c)            addi $5, $0, 3               # counter %
 8 : 20a5ffff; % (20) loop2:   addi $5, $5, -1              # counter - 1 %
 9 : 34a8ffff; % (24)            ori $8, $5, 0xffff           # zero-extend: 0000ffff %
 A : 39085555; % (28)            xori $8, $8, 0x5555          # zero-extend: 0000aaaa %
 B : 2009ffff; % (2c)            addi $9, $0, -1              # sign-extend: ffffffff %
 C : 312affff; % (30)            andi $10, $9, 0xffff         # zero-extend: 0000ffff %
 D : 01493025; % (34)            or $6, $10, $9               # or: ffffffff %
 E : 01494026; % (38)            xor $8, $10, $9              # xor: ffff0000 %
 F : 01463824; % (3c)            and $7, $10, $6              # and: 0000ffff %
10 : 10a00001; % (40)            beq $5, $0, shift            # if $5 = 0, goto shift %
11 : 08000008; % (44)            j loop2                      # jump loop2 %
12 : 2005ffff; % (48) shift:   addi $5, $0, -1              # $5 = ffffffff %
13 : 000543c0; % (4c)            sll $8, $5, 15               # <<15 = ffff8000 %
14 : 00084400; % (50)            sll $8, $8, 16               # <<16 = 80000000 %
15 : 00084403; % (54)            sra $8, $8, 16               # >>16 = ffff8000 (arith) %
16 : 000843c2; % (58)            srl $8, $8, 15               # >>15 = 0001ffff (logic) %
17 : 08000017; % (5c) finish: j finish                      # dead loop %
18 : 00004020; % (60) sum:      add $8, $0, $0               # sum %
19 : 8c890000; % (64) loop:     lw $9, 0($4)                 # load data %
1A : 20840004; % (68)            addi $4, $4, 4               # address + 4 %
1B : 01094020; % (6c)            add $8, $8, $9               # sum %
1C : 20a5ffff; % (70)            addi $5, $5, -1              # counter - 1 %
1D : 14a0fffb; % (74)            bne $5, $0, loop             # finish? %
1E : 00081000; % (78)            sll $2, $8, 0                # move result to $v0 %
1F : 03e00008; % (7c)            jr $ra                       # return %
END ;
```

4、sc_datamem.mif 文件内所存的数据存储器初始值如下：

```
DEPTH = 32;                 % Memory depth and width are required %
WIDTH = 32;                 % Enter a decimal number %
ADDRESS_RADIX = HEX;     % Address and value radixes are optional %
```

```
DATA_RADIX = HEX;          % Enter BIN, DEC, HEX, or OCT; unless %
CONTENT                    % otherwise specified, radixes = HEX %
BEGIN
14 : 000000A3;             % (50) data[0] %
15 : 00000027;             % (54) data[1] %
16 : 00000079;             % (58) data[2] %
17 : 00000115;             % (5C) data[3] %
END ;
```

三、仿真验证

补充完整后用提供的 sc_computer_test_wave_01.vwf 对 CPU 模块执行示例程序过程进行仿真，能够得到如下仿真结果即为完成。通过仿真波形，可查看到示例程序的运行过程符合程序设想，PC 指针、指令数据、结果数据等均正确：

图 3 为总图，程序执行到最后，停在最后一条死循环跳转指令处，inst=08000017，pc=0000005c。

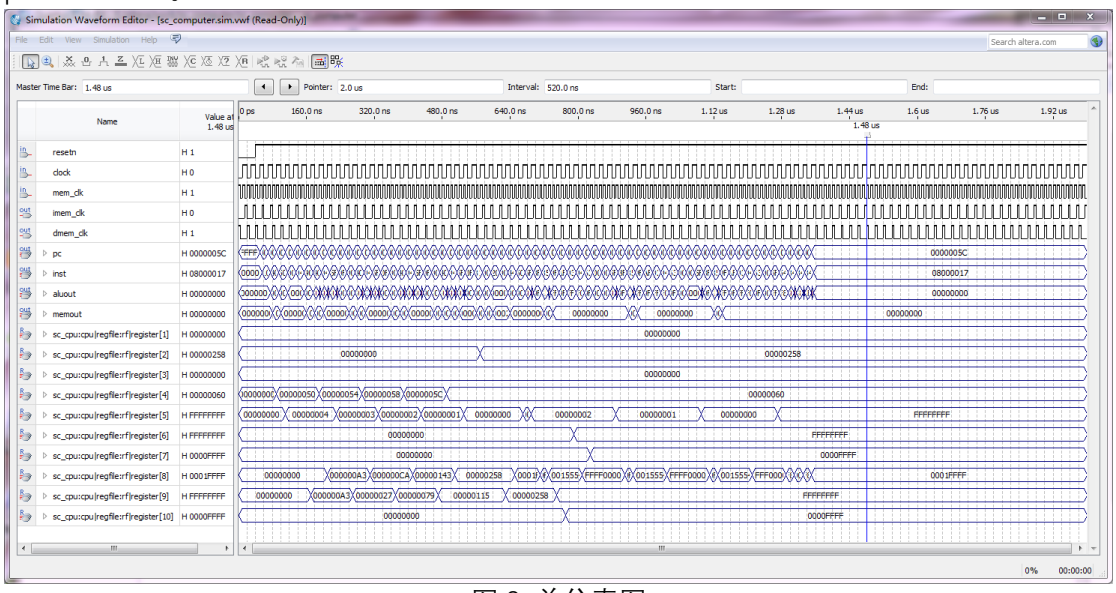

图 3 总仿真图

放大后从前至最后的细节如图4至图9，可以看到每条指令执行的顺序过程及寄存器、ALU输出等数据变化。

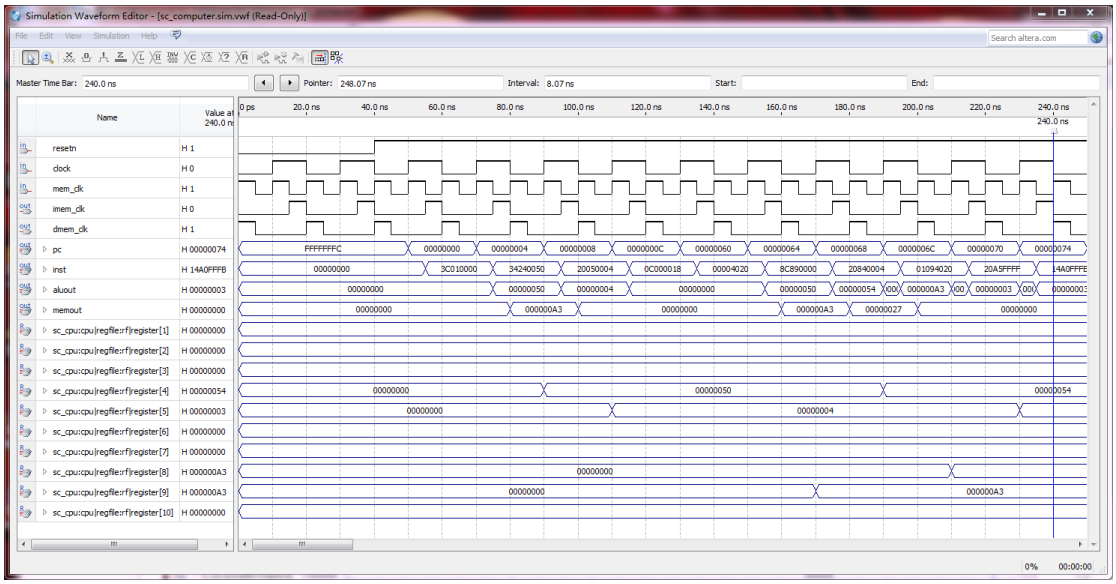

图 4 仿真图细节 1

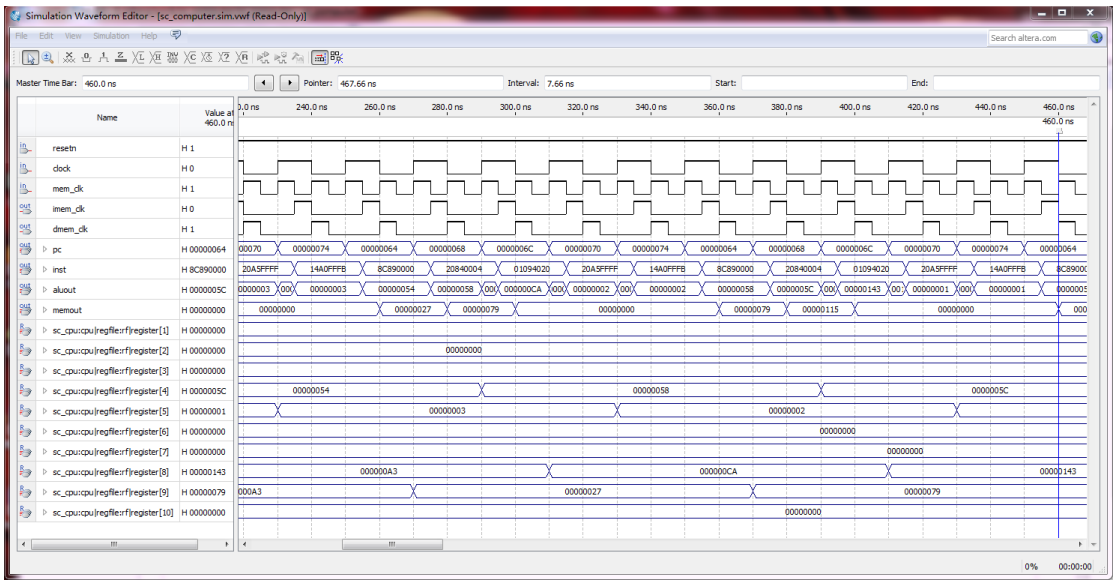

图 5 仿真图细节 2

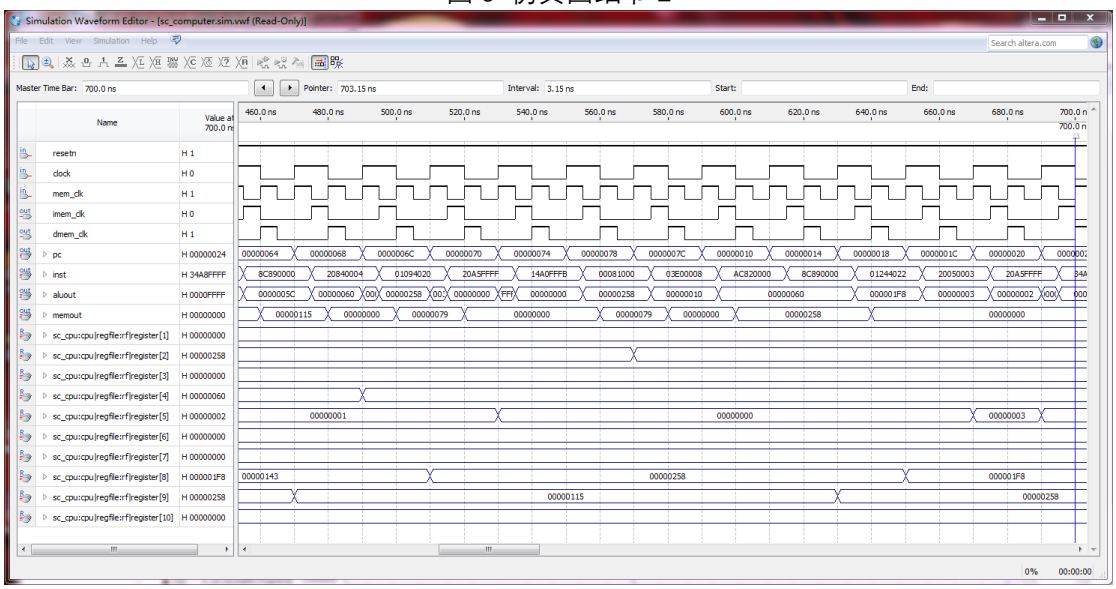

图 6 仿真图细节 3

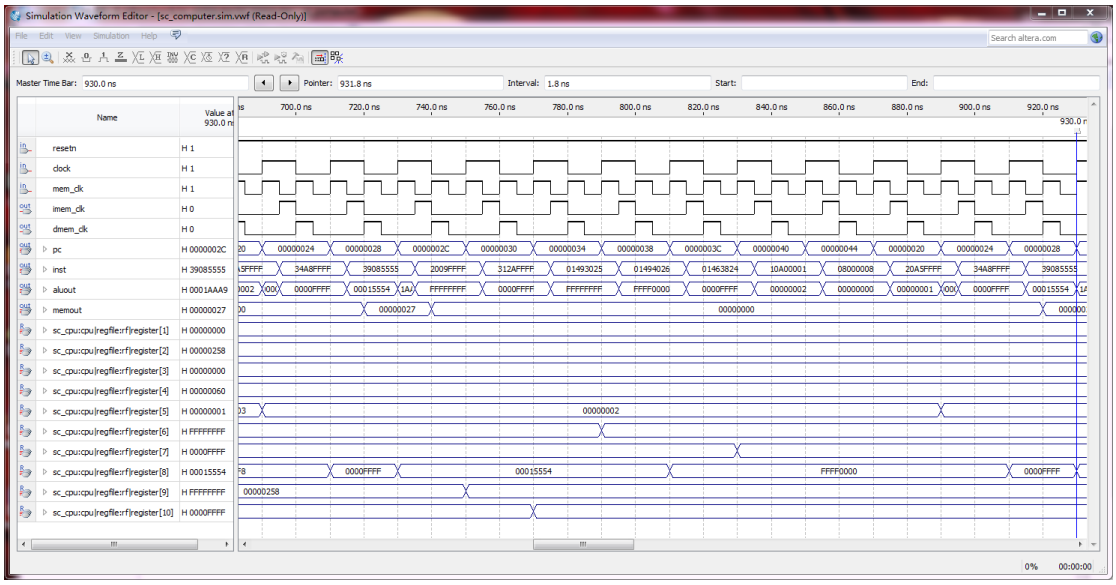

图 7 仿真图细节 4

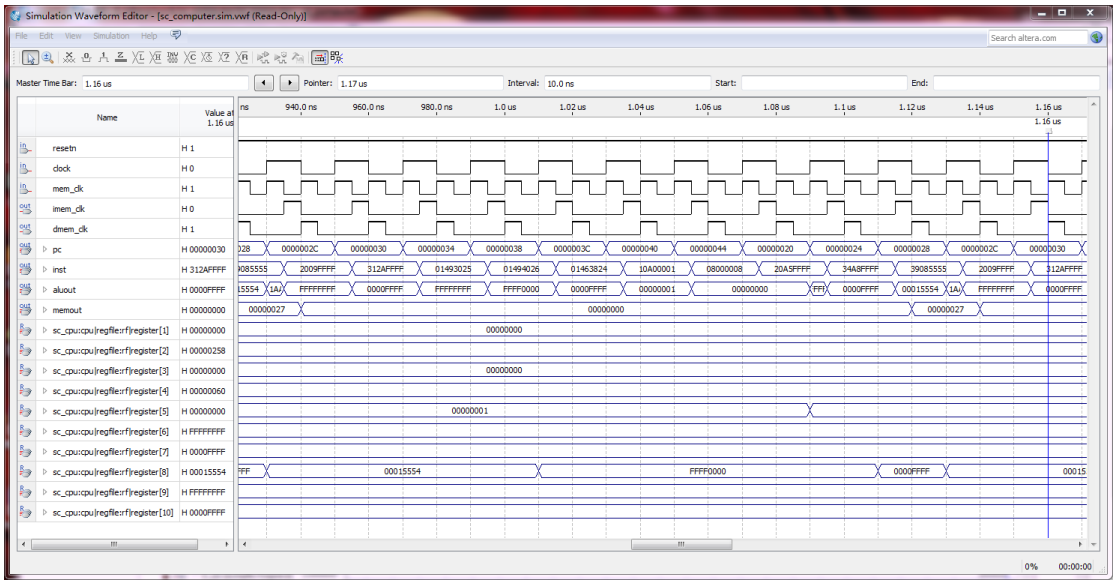

图 8 仿真图细节 5

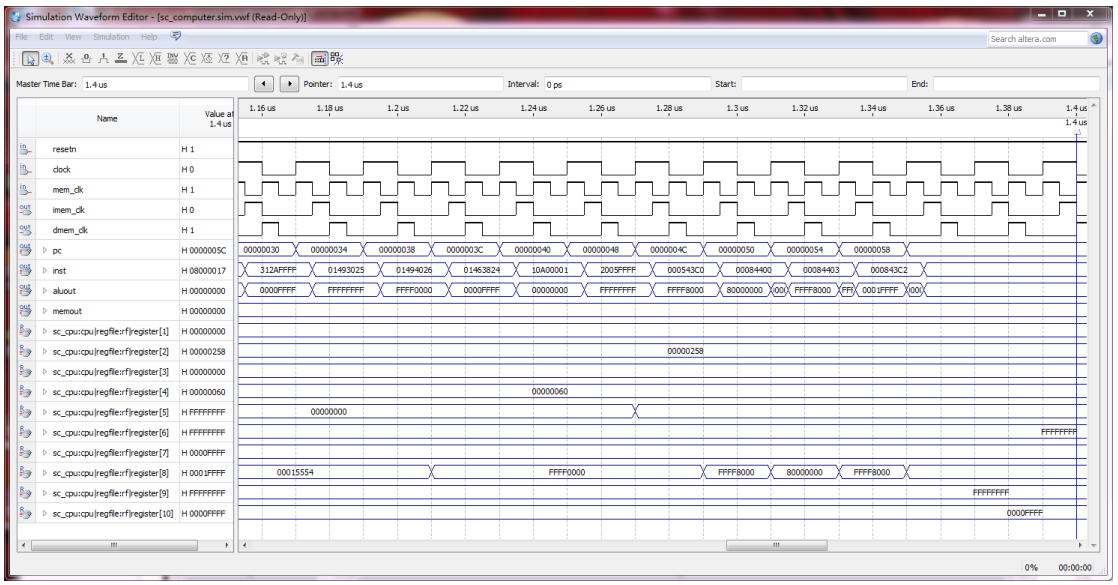

图 9 仿真图细节 6

四、单周期 CPU 设计实验中的相关问题解答

【问题 1】如何在设计中加入 Quartus II 中 Altera 提供的逻辑器件库中的模块？

【解答】如果从最基本的逻辑单元来设计和构建一个大型的逻辑系统，事无巨细的逻辑设计会使工作量巨大，设计效率不高。Quartus II 中 Altera 提供了较为丰富的基本逻辑宏模块设计库，包括如加法器、乘法器、除法器、log 运算器、sqrt 平方根运算器等 ALU 部件，以及多种接口部件等（详见 Quartus II 使用文档）。

另外，Altera 提供有嵌入式的 Nios II 软 CPU 核，可以在 FPGA 中构建 CPU 核，甚至多核，用以实现功能更强大的 SOPC（System On a Programmable Chip）系统。

实验中所设计的指令 ROM 和数据 RAM 存储器，可以利用 Quartus II 中提供的 ROM 和 RAM 宏模块进行设计。由厂家提供的这些宏模块设计，应该是根据其 FPGA 器件的特点，所实现的最优设计。

Quartus II 中所提供的 ROM 和 RAM 设计，均为同步器件，即包含有同步信号，这和

单周期 CPU 原理设计中的异步 ROM 和异步 RAM 是有所不同的。采用具有同步信号的设计，能够使信号的变化和传输（延迟）过程等更加明确，增强了电路设计的确定性。

对这些宏模块的应用，在 Quartus II 中可以进行配置并选择生成为 Verilog 的.v 文件的方式，该.v 文件中包含有对所选用宏模块的模块定义，类似于 C 语言中对过程函数的原型声明，这样，其它模块就可以利用该宏模块声明来实例化其具体应用。在设计中利用 Quartus II 所提供的逻辑器件，其方法及步骤如下：

1、在“File”菜单中选择”New…”菜单项，出现如图 10 所示界面，选择”Block Diagram/ Schematic File”，然后点击“OK”按钮，出现图 11 所示 Block1.bdf 界面。

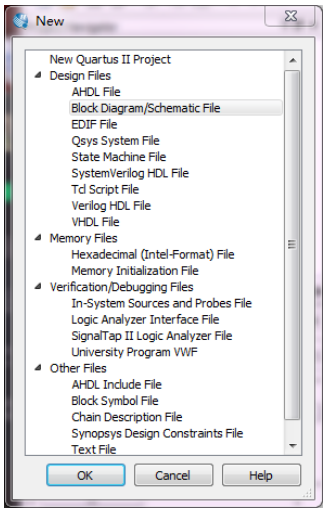

图 10. 选择建立模块文件界面

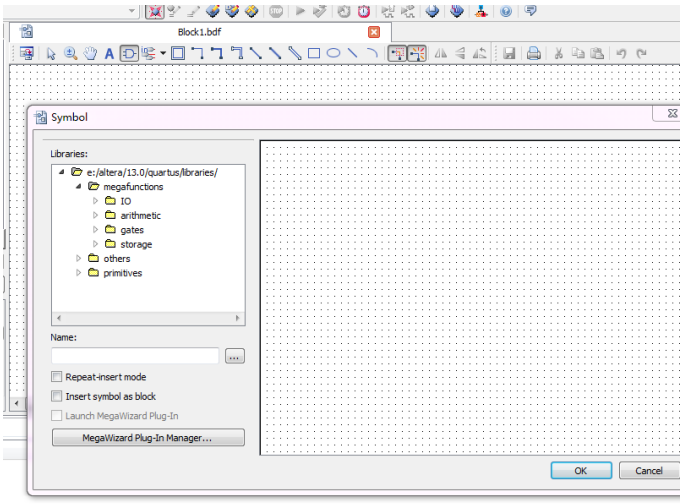

图 11. 选择库中的宏模块界面

2、鼠标“双击”图 11 所示 Block1.bdf 界面视图，出现如图中所示的“Symbol”界面，在其左上方的库模块列表中，就可以选择所需的器件模块。

3、在图 11 所示“Symbol”界面中，选择实验中用到的“megafunctions”下“storage”中的“lpm_rom”或“lpm_ram_dq”模块。只读 ROM 相对简单，以同步 RAM 为例，选择“lpm_ram_dq”后的界面如图 12 所示，右侧显示了该模块的初步设计功能框图，接下来可以点击左下角的“Megawizard Plug_In Manager…”按钮对其特性进行配置。

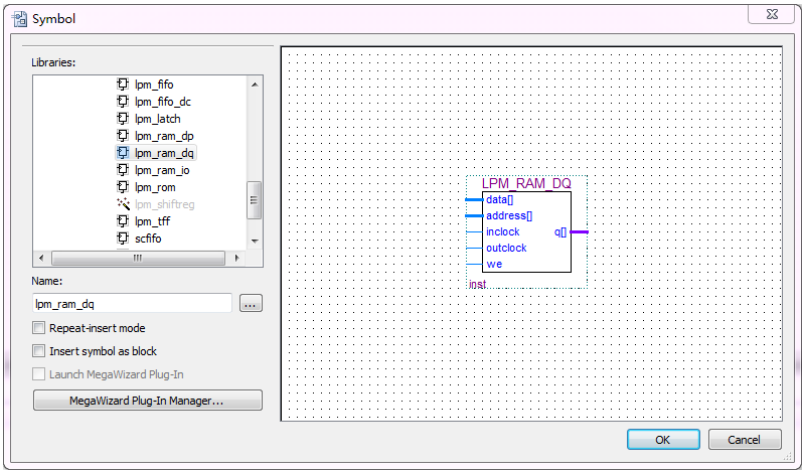

图 12.选择库中“lpm_ram_dq”宏模块

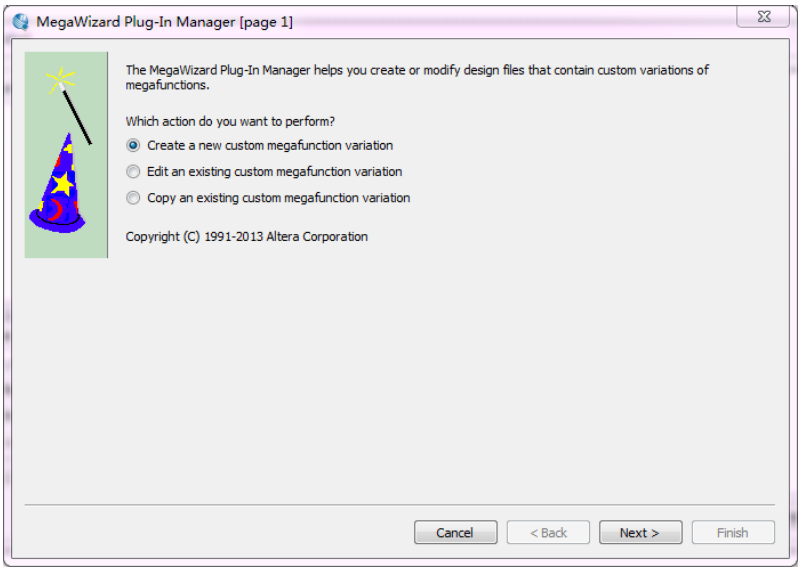

图 13 选择新建一个用户例化宏模块

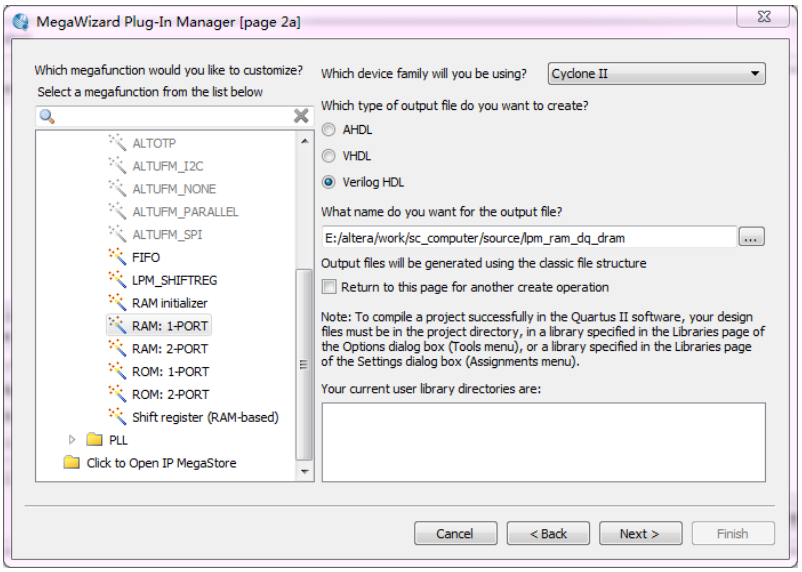

图 14.选择输出文件形式及其存放位置

4、在图 12 所示界面，点击左下角“Launch MegaWizard Plug-In…”按钮，点击“Next”按钮，在图 13 所示界面选择新建一个用户例化宏模块，进入对该 RAM 的配置过程。

5、在如图 14 所示的“MegaWizard Plug-In Manager”界面中，选择 “Verilog HDL”文件输出形式，指定模块由哪个系列 FPGA 实现，并指定自己接下来要生成的 .v 设计文件的存放位置及文件名。建议和自己设计的其它源文件放在同一目录下，以方便管理。然后点击“Next”。

6、在接下来的配置过程当中，可按实验中的应用需要，依次进行属性选项的配置。如图 15 至图 18 所示，依次设置 RAM 位宽为 32bits，RAM 容量为 32words；为便于代码在不同 FPGA 器件间移植，“memory block type”选择“Auto”；选择单一时钟，即 Input 和 Output 逻辑采用相同的时钟；不选 “‘q’output port”，即 RAM 输出不需要加入寄存器缓冲输出级，否则会需要额外的一个时钟周期后才能输出 RAM 中的数据；指定实验中数据 RAM 的初始化文件“sc_datamem.mif ”；最后选择生成输出.v 模块定义文件，以及器件的工作时序波形文件。

7、“Finish”后生成的“lpm_ram_dq_dram.v”文件中，就包含了对所建 RAM 模块的定义，在其它模块中就可以例化应用。同学可以打开该自动生成文件进行查看其结构并确认，也只有该“lpm_ram_dq_dram.v”是工程文件中所必需的。

8、由同学自行学习建立自己的“指令 ROM”存储器。

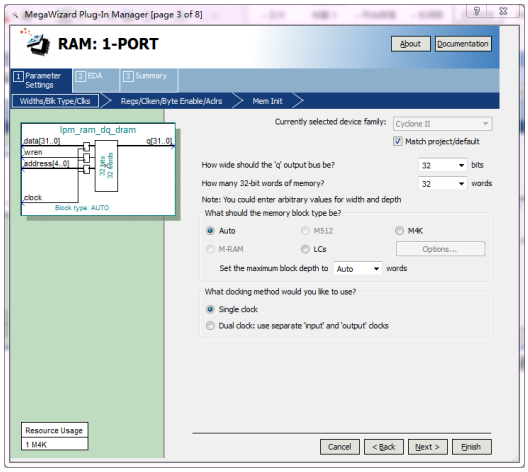

图 15. 设置 RAM 位宽、容量及时钟

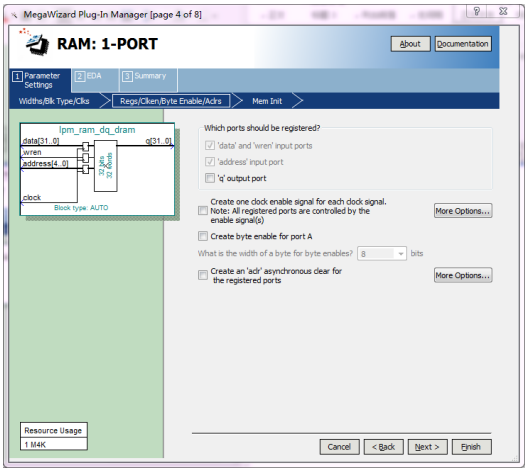

图 16. RAM 输出级不需要寄存器缓冲结构

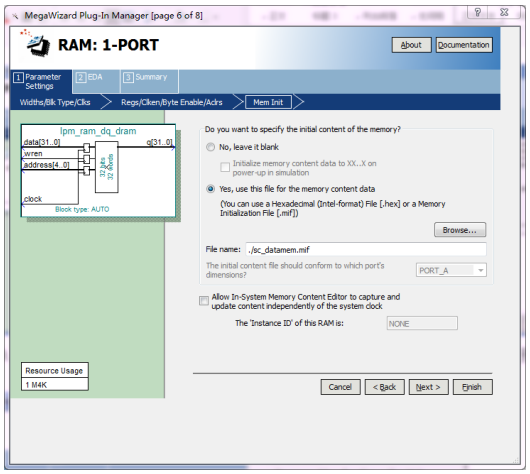

图 17. 设置 RAM 初始化数据文件

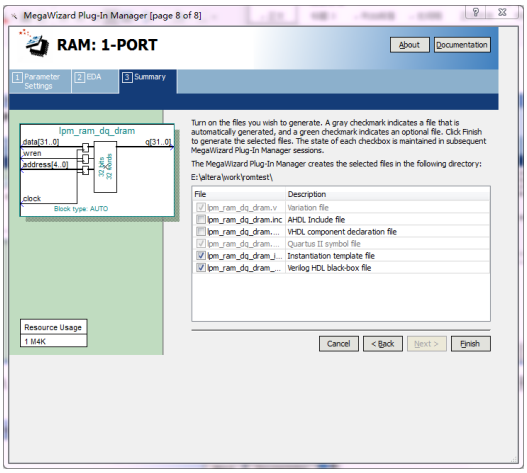

图 18. 选择输出.v 文件及器件波形图文件

【问题 2】 单周期 CPU 设计实验中除了一个基本的 clock 时钟信号外，为什么还设计有一个 2 倍频于 clock 信号、标记为 mem_clk 的时钟信号?

【解答】这是每个指令周期内，对各逻辑功能部件的工作时间顺序进行控制的需要。

从单周期 CPU 的设计原理上，如果指令 ROM 和数据 RAM 都为异步器件，即只要收到地址就开始送出数据，则在实现中只需要一个 clock 时钟信号控制 PC 的变化、以及整条指令的执行时间（机器周期）就够了。

但在 DE1-SOC 实验板上 Altera 的 Cyclone V 系列 FPGA 器件中，只提供同步ROM 和同步 RAM 宏模块。在【问题 1】的解答中所建立的同步 ROM 和同步 RAM， 其读写时序如图 19 和图 20 所示。

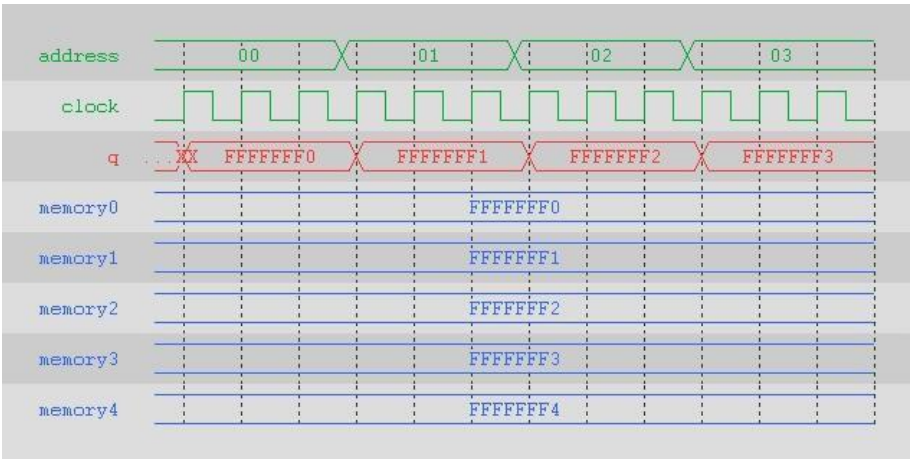

图 19. 同步 ROM 和同步 RAM 的“读”操作工作时序

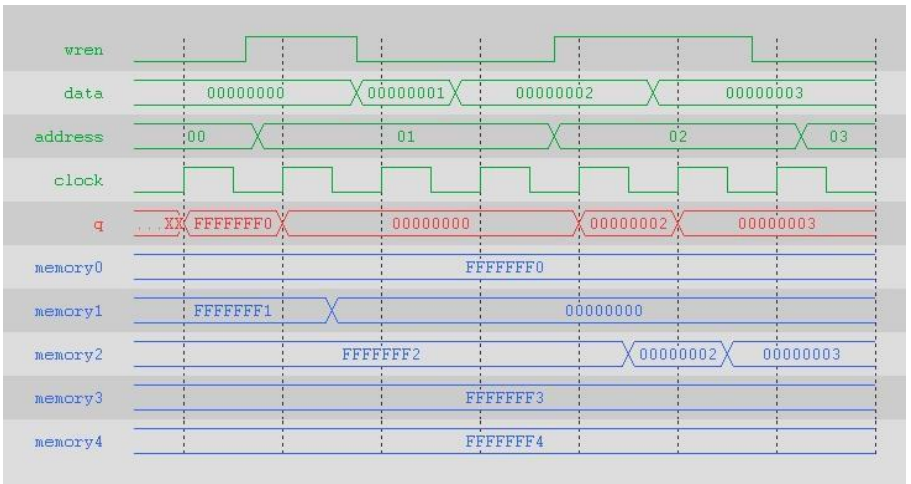

图 20. 同步 RAM 的“写”操作工作时序

考察图 19 所示同步读时序，当同步 ROM 或 RAM 接收到数据地址时，不同于异步器件在经过一段组合电路的反应延迟后送出数据，同步器件需要等到同步时钟信号 clock 上升沿时才真正送出数据（这就可以使延迟时间不尽相同的多个器件在输出级达成一致）。

考察图 20 所示同步写时序，同步 RAM 在接收到地址后，在 wren 信号高有效期间，需要等到同步时钟 clock 上升沿，才开始写动作，并将要写入的新数据马上输出到 q 输出端，即在写时可以同时读输出，满足图 19 的读时序，并在同步时钟 clock 的下降沿将数据真正写入到存储器中。

对于单周期 CPU 指令执行的一系列过程（5 个大步骤）：

① 取指令：PC 在时钟上沿改变新值，按改变后并稳定的新 PC 值作为指令地址，访问指令 ROM 取出指令；

② 译码：指令的一部分交由控制器 CU 进行译码，另一部分至寄存器堆取操作数；

③ 执行：ALU 等部件执行相应操作；

④ 访问数据 MEM： 读或写数据存储器；

⑤ 回写寄存器：回写结果至目的寄存器。

以上各步骤在逻辑上是顺序执行的，在时间上是有先后顺序的，因此，对于采用有同步信号的指令同步 ROM、数据同步 RAM，在一个 CPU 指令周期内，需要安排更精细的时钟信号来协调它们的顺序工作过程。

为此，实验中设计了 2 路时钟信号，一路为表征机器周期的 clock 时钟信号，另一路为 clock 时钟信号 2 倍频的 mem_clk 时钟信号。事实上，在实现中，clock 是mem_clk 信号的 2 分频信号，分频比倍频在电路上要容易实现的多。信号波形如图21 所示。

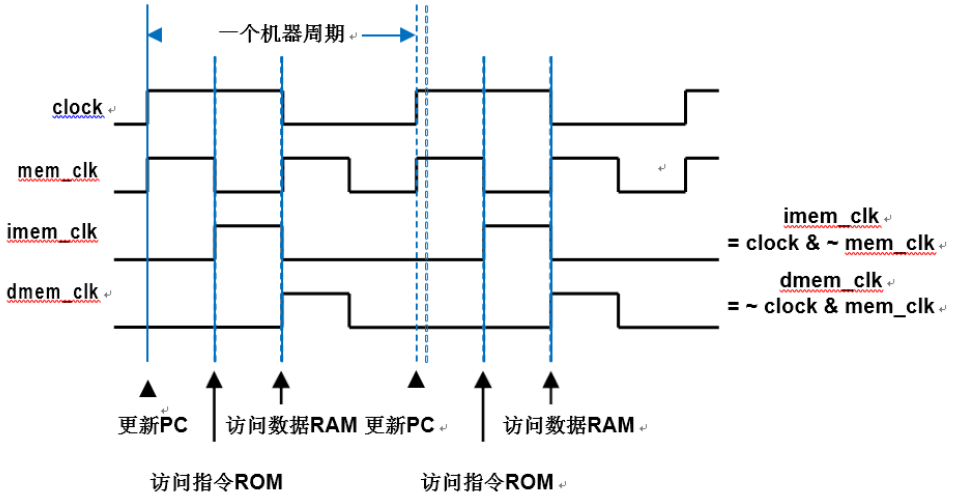

图 21. 一个机器周期内各执行步骤的执行触发顺序

如图 21 所示，由输入信号 clock 和 mem_clk 可通过组合逻辑生成 4 个节拍信号，实验中根据时序需要，选取图中标注为 imem_clk、dmem_clk 的两个节拍分别作为指令 ROM 和数据 RAM 的同步时钟。这样，CPU 的工作就在一个机器周期内，有条不紊地顺序执行。各步骤上升沿间的时间间隔，就是留给各阶段组合逻辑的信号传输及延迟时间。

利用机器周期和更高频率（如 8 倍频）信号的组合，可以将一个过程分为更多的节拍（如 8 个节拍），从而安排更多功能部件的协调有序执行。