Unicore32模拟器整体设计文档（草案）

李春奇　　彭焯　　华连盛　　王衍

December 30, 2010

模拟器架构主要分为三个部分：进程模块、CPU模块和调试模块

1　进程模块

进程模块的接口声明在文件process.h中，实现在process.c中；进程模块包含的子模块有：内存管理模块、ELF文件解析模块。

进程模块的功能描述：进程模块的功能是建立一个进程，包括进程自己内存空间的初始化和维护、进程代码和数据的载入、程序入口点等。

进程模块的接口如下：

```
1    extern PROCESS* proc_initial(char* filename);
2    extern int proc_destroy(PROCESS* proc);
```

功能分别为建立进程和销毁进程，其中filename为进程的物理文件名（ELF可执行文件），proc为需要销毁的进程。

1.1　内存管理子模块（memory.h memory.c）

1. 内存管理子模块分为两个部分：栈空间管理和内存空间管理
2. 栈空间管理
3. 内存空间管理

1.2　ELF文件解析模块（ELF_parser.h ELF_parser.c）

ELF文件解析模块主要是用来解析ELF文件，建立对应的Segment，并将栈作为Segment统一管理。

2 CPU模块

CPU模块的接口声明在文件CPU.h中，实现在CPU.c中；CPU模块包含的子模块有：寄存器堆模块、流水线模块、cache模块

2.1 寄存器堆模块

寄存器堆模块维护的是每个CPU对应的寄存器堆，主要需要维护CMSR的N、Z、C、V的读和写。

2.2 cache模块

Cache模块需要实现如下功能:

1. Cache的读写
2. Cache与内存的一致性问题，即对于写脏的内存的Cache更新策略。
3. Cache的Miss和Hit机制
4. Cache的写回机制

2.3 流水线设计

流水线设计是模拟器设计的重点，下面对流水线进行详细的描述。

2.3.1 流水结构

五级流水:
取指（IF）--译码（读寄存器）（ID）--ALU计算（EX）--访存（MEM）--写回（WB）

2.3.2 需要解决的冒险

1. 结构冒险：通过指令cache和数据cache分离的Havard结构解决
2. 数据冒险:

   数据转发机制解决一部分冒险问题

   但是“加载-使用型数据冒险”需要加一个气泡

3. 控制冒险:

分支控制导致的冒险: 用分支预测的方法来解决:

分支预测的策略:a.B指令一定预测跳转，b.其他条件跳转指令预测不跳转

2.4 各级流水的设计和接口规范

2.4.1 IF（Instruction Fetch）

输入: 指令地址
输出: struct
实现: 直接对cache进行访问，分为命中和未命中两种情况

2.4.2 ID（Instruction Decode）

输入: IF的输出
输出: struct
实现: 对指令进行译码，获得对应寄存器的值，寄存器的编号，操作类型、操作数、移位立即数的值等等并保存在结构体中

2.4.3 Ex（Execuation）

输入: ID的输出
输出: struct
实现: 根据指令的类型进行相应的运算，对于R型指令，写回在此处进行（即在此处进行数据转发）；对于分支指令，分支预测在此处进行处理。

2.4.4 Mem（Memory）

输入: Ex的输出
输出: struct
实现: lw指令的写回在此处实现，其他指令到此处均已执行完毕

2.4.5 WB（Write Back）

所有指令均已经执行完毕，此模块只做一些初始化工作。

2.5 各模块调用次序及相关结构的定义

各模块在逻辑上是并行的，在实现上是从后向前调用，遇到需要插入气泡的情况直接阻塞前边的各级流水

上文所述的结构:

```
typedef struct{
    uint32_t inst_addr;
    uint32_t inst_code;
    int inst_type;
    int opcodes;
    uint32_t Rn, Rd, Rs, Rm;
    int imm;
    int shift;
    int rotate;
    int cond;
    int high_offset, low_offset;
    int S, A, P, U, B, W, L, H;
    uint32_t cur_inst_PC;
    uint32_t addr;
}PIPLINE_DATA;
```

流水线的数据结构如下:

```
typedef struct{
    int block;//1 means pipline block, 0 mean the opposite
    int block_reg;
    PIPLINE_DATA* pipline_data[PIPLINE_LEVEL];
    char pipline_info[PIPLINE_LEVEL][200];
    PROC_STACK* stack;
    REGISTERS* regs;
    CACHE *i_cache, *d_cache;
    PROCESS* proc;
    int drain_pipline;
    int pc_src;
    int ex_begin;
}PIPLINE;
```