|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
|  | | | | | | |  |
| МИНОБРНАУКИ РОССИИ | | | | | | |  |
| Федеральное государственное бюджетное образовательное учреждение  высшего профессионального образования  **«МИРЭА – Российский технологический университет»**  **РТУ МИРЭА** | | | | | | |  |
|  | Институт информационных технологий (ИТ) | | | | | | |
|  | Кафедра инструментального и прикладного программного обеспечения (ИППО) | | | | | | |
| **Отчет по лабораторной работе №1**  По дисциплине «Разработка ПАОИ и АС»  Тема: «Разработка модели процессорного ядра программно-аппаратного комплекса» | | | | |
|  | | | | |
|  | | | | |
| Выполнил студент группы ИКМО-01-19 | | | Косиков М.И. | |
| Преподаватель | | | Тарасов И.Е. | |
|  | |  | |  | |
|  | |  | |

Москва 2019

**Цель работы**: освоение маршрута проектирования процессорных ядер на уровне системного моделирования путем разработки модели процессорного ядра на языке программирования высокого уровня.

**Введение**

При проектировании процессоров важную роль играет предварительное моделирование. При использовании программы, имитирующей работу будущего процессора, становится возможным проверить результаты выполнения на этом процессоре программ и убедиться, что для этого имеются достаточные ресурсы, набор поддерживаемых команд, а алгоритм выполняется корректно и за приемлемое число шагов. Предварительное моделирование, выполняемое с помощью широко распространенной компьютерной техники, может быть организовано быстрее и не требует безвозвратных финансовых потерь по сравнению с проектированием процессора в виде интегральной микросхемы.

Такое моделирование процессора называется *эмуляцией*.

«Имитация функционирования одного устройства посредством другого устройства или устройств вычислительной машины, при которой имитирующее устройство воспринимает те же данные, выполняет ту же программу и достигает того же результата, что и имитируемое» [из п. 53 табл. 1 ГОСТ 15971-90].

«Эмуляция – комплекс программных, аппаратных средств или их сочетание, предназначенное для копирования функций одной вычислительной системы на другой, отличной от первой, вычислительной системе таким образом, чтобы эмулированное поведение как можно ближе соответствовало поведению оригинальной системы. Примечание - Целью эмуляции является максимально точное воспроизведение поведения в отличие от разных форм моделирования, в которых имитируется поведение некоторой абстрактной модели. Например, моделирование физического процесса или явления не является эмуляцией» [из п. 3.7 ГОСТ Р 57721-2017]

Эмуляция может выполняться на нескольких уровнях, различающихся детализацией моделирования. При необходимости определить поведение процессора при выполнении алгоритмов может быть выполнено моделирование на уровне программной модели.

Пример программной модели процессора может выглядеть, как показано на рис. 1.

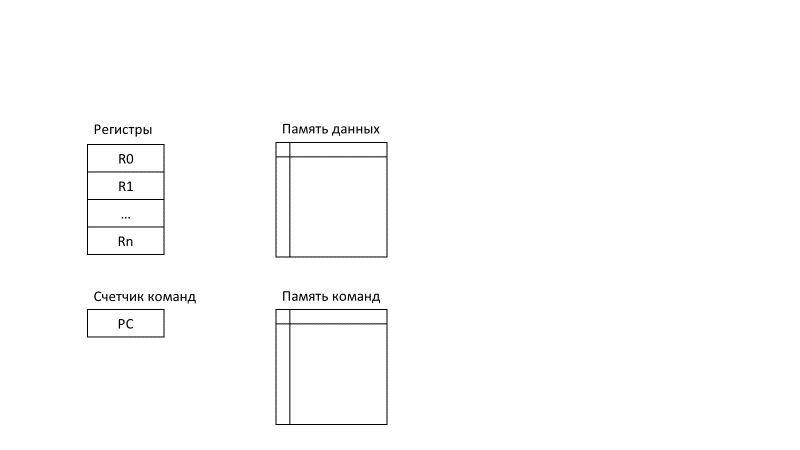


Рис. 1. Программная модель процессора

В модели показаны базовые компоненты процессора. Регистр PC (Program Counter) называется также «счетчик команд». Этот регистр содержит адрес памяти, из которого в данный момент исполняется команда. Память команд и данных может быть физически разделена (т.е. процессор не может выполнять команды из памяти данных). Это решение соответствует *гарвардской архитектуре* процессора. Если же команды и данные хранятся в одной и той же памяти, говорят об *архитектуре фон Неймана*.

Спецификой памяти является ограниченное количество чисел, которые можно читать из нее в каждый момент времени (обычно это одно число, однако существует и многопортовая память). Поэтому для выполнения вычислений процессор обычно загружает данные из памяти во внутренние узлы для хранения чисел – регистры данных. Существует множество вариантов организации регистров данных – их количества, разрядности, допустимых операций между этими регистрами и т.д.

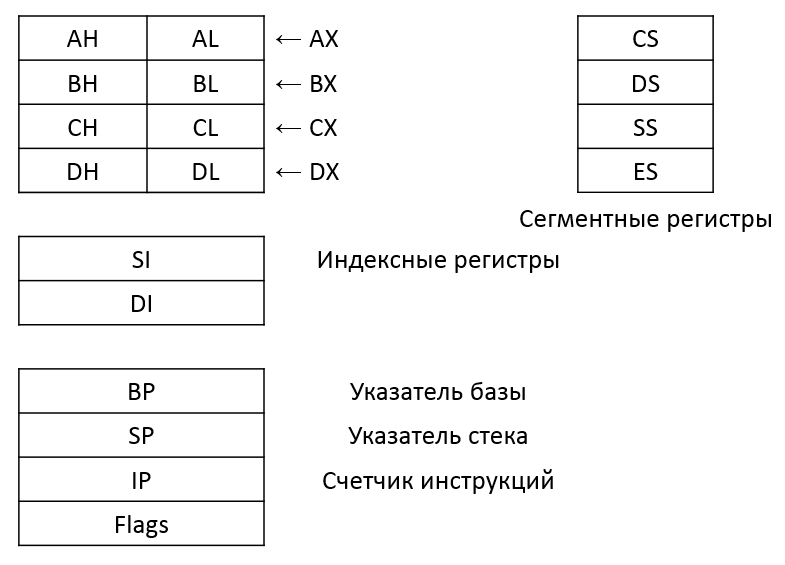


Рис. 2. Регистровая модель 16-разрядного процессора i8086.

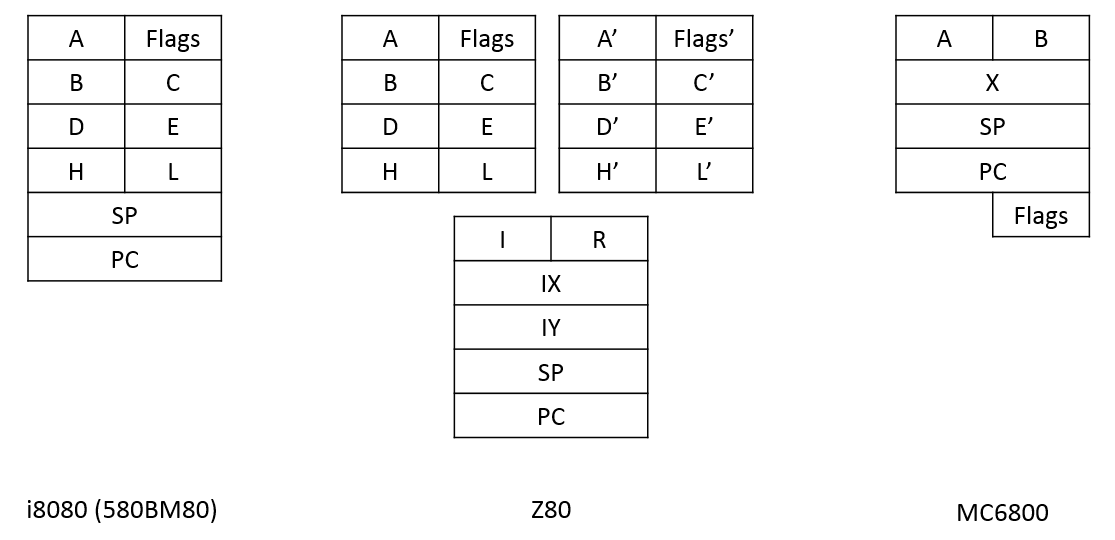


Рис. 3. Регистровые модели процессоров i8080, Z80 и MC6800

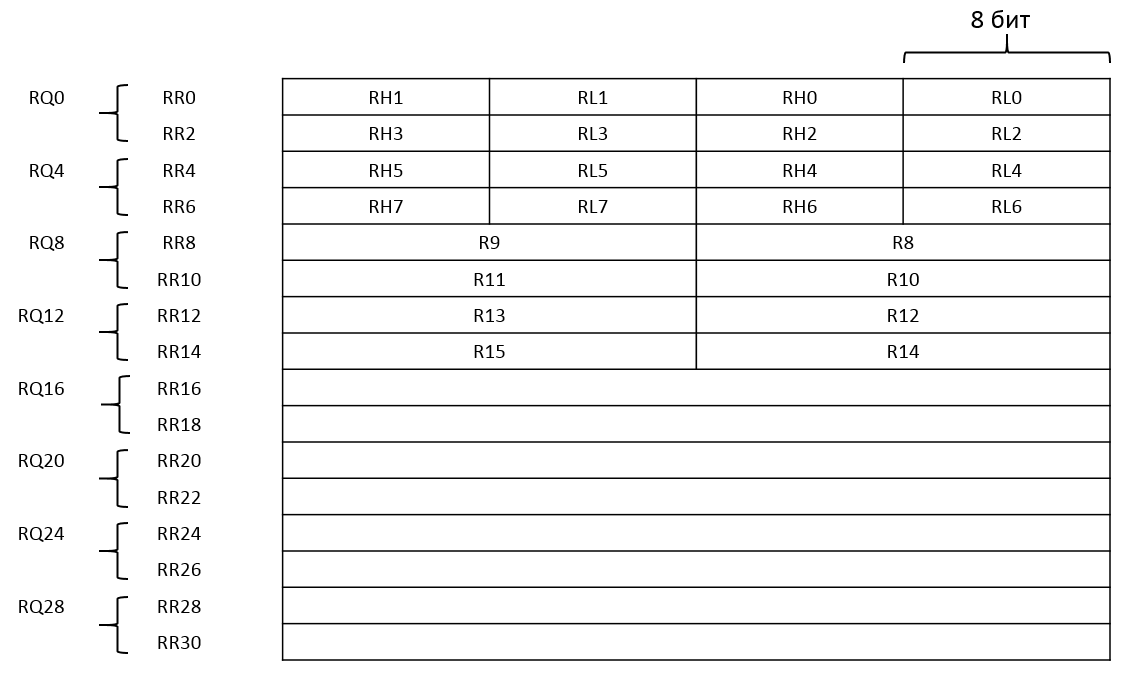


Рис. 4. Регистры общего назначения процессора Z80000 (1986 г.)

Моделирование процессора производится путем программной имитации действий, выполняемых аппаратными компонентами процессора в ходе его работы. При этом игнорируются технические детали работы таких схем – например, для имитации чтения данных из памяти достаточно прочитать значение из массива, который имитирует память, подключенную к процессору. Состояние управляющих сигналов памяти, времена распространения сигналов и прочие существенные для микроэлектроники параметры не учитываются. Подразумевается, что при правильном подключении памяти к процессору операции чтения и записи будут производиться корректно.

В простейшей модели типичный процессор циклически выполняет следующие действия:

1. Чтение команды из ячейки памяти, адрес которой содержится в регистре PC.

2. Выполнение действий, кодируемых этой командой, включая обязательное вычисление нового значения для регистра PC.

В простейшем случае фрагмент программы, имитирующей работу процессора, будет выглядеть следующим образом:

int cmd;

int pc;

int cmem[1024];

int dmem[1024];

while(1)

{

cmd = cmem[pc];

// здесь впоследствии необходимо описать

// модель выполнения принятой команды

pc = pc + 1;

}

В этом примере имеются упрощения:

– не предусматривается выход из цикла, что в целом соответствует нормальной работе процессора, но неприемлемо для программы, которая в таком виде будет работать бесконечно;

– использован фиксированный размер массива, имитирующего память команд процессора и фиксированные целочисленные типы для регистра PC и ячеек памяти;

– отсутствует имитация исполнения команд.

Для имитации выполнения команд необходимо выполнить планирование системы команд. Команда, в примере выше записанная в переменную cmd, обычно содержит отдельные разряды, кодирующие выполнение тех или иных действий. Для показанного на рис. 1 примера можно представить простой набор команд, задаваемые разрядами числа, как показано на рис. 2. Сгруппированные двоичные разряды называют также полями, т.е. разряды с 3 по 0 можно называть «поле второго операнда».

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Разряды двоичного представления команды | 31-28 | 27-12 | 11-8 | 7-4 | 3-0 |
| Название | cmdtype | literal | dest | Op1 | Op2 |
| Описание | Тип команды | Непосредственное значение («литерал») | Номер регистра для записи результата | Номер регистра – первого операнда | Номер регистра – второго операнда |

Рис. 5. Формат команды для примера процессора

Поскольку цифровая схема не ограничена использованием форматов чисел, принятых в программировании, двоичное представление команды может использоваться внутренними схемами процессора как набор независимых сигналов. Например, если в процессоре имеется 16 регистров, то номер регистра может быть задан в 4-разрядном двоичном числе. Для кодирования команды следует задать номера ее операндов, регистр-получатель результата, а также тип команды. В примере на рис. 2 для типа команды отведено также 4 разряда, поэтому возможно кодировать 16 вариантов команды. Например, если принять, что сложение регистров имеет код команды 1, то команда для сложения регистров 3 и 4, помещающая результат в регистр 7, будет выглядеть следующим образом:

Команда 1 (сложение)

0001\_????????\_????????\_0111\_0011\_0100

Регистр-получатель 7

Регистры-операнды 3 и 4

Знаками вопроса показаны разряды, которые не используются в команде и могут принимать любые значения.

Поскольку для примера выбраны 4- и 16-разрядные поля команды, удобно задавать код команды в шестнадцатеричном формате. Чтобы поместить показанную команду в память, достаточно выполнить присваивание:

cmem[0] = 0x10000734

Тогда при pc = 0 процессор прочитает из памяти эту команду, поля которой будут кодировать требуемые действия. Удобно использовать вспомогательные переменные, в которые можно записать значения отдельных полей команды. Чтобы выделить из команды отдельные разряды, можно использовать операцию «поразрядное И». В результате поразрядного «И» каждый бит результата становится равным 1, только если соответствующие биты обоих операндов равны 1. Например, если выполнить операцию cmd & 1, то результатом будет число 0 или 1, в зависимости от того, был ли равен 1 самый младший разряд двоичного представления переменной cmd. Чтобы выделить разряды с 3 по 0 и записать результат во вспомогательную переменную op2, необходимо выполнить операцию

Op2 = cmd & 15

Число 15 имеет двоичное представление 1111, т.е. содержит единицы в четырех младших разрядах.

Для того, чтобы выделять поля, находящиеся в других разрядах, следует сначала выполнить сдвиг двоичного представления, чтобы требуемый фрагмент команды оказался в младших разрядах. Например, для выделения поля первого операнда необходимо выполнить операцию

Op1 = (cmd >> 4) & 15

Сначала оператор сдвига перемещает разряды 7 – 4 в позиции 3 – 0, затем оператор & выделит 4 младших разряда.

Аналогично следует организовать выделение остальных полей команды.

Далее следует организовать моделирование выполнения принятой команды. Поскольку номера регистров, участвующих в команде, представляют собой индексы массива регистров, имитацию сложения можно выполнить следующим образом:

Switch (cmdtype)

{

Case 1 : reg[dest] = reg[op1] + reg[op2]; break;

Default : break;

}

Подобным же образом можно описать другие команды, выполняющие арифметическую или логическую операцию над входными операндами op1 и op2. К числу операций, реализуемых в процессорах, обычно относятся:

– вычитание;

– поразрядное И (and);

– поразрядное ИЛИ (or);

– поразрядное ИСКЛЮЧАЮЩЕЕ ИЛИ (xor);

– инверсия одного из операндов (not);

– сдвиги влево и вправо.

Операция умножения, очевидная для арифметики, должна использоваться в процессорах с осторожностью. Схема умножения двоичных чисел занимает существенно больше ресурсов по сравнению с другими операциями, поэтому ряд процессорных ядер не имеет команды умножения, или реализует ее последовательными шагами по схеме «сложение со сдвигом» (аналогично умножению в столбик, записанному для двоичных чисел).

Операция целочисленного деления не имеет общепринятой схемы на базе логических элементов (могут существовать реализации деления на базе таблиц, однако они существенно ограничены по разрядности операндов). Как правило, используется команда «шаг деления», вычисляющая один бит результата по алгоритму, аналогичному делению в столбик.

Кроме представленного формата команды, для моделирования процессора следует реализовать как минимум два дополнительных варианта.

Первый из этих вариантов – так называемая непосредственная адресация (Indirect). Если представить программу, выполняющую сложение чисел 2 и 3, то из рассмотренного ранее формата команд видно, что попытка указать эти числа в полях op1, op2 приведут к использованию регистров 2 и 3, но не чисел 2 и 3. Поэтому необходимо добавить поле команды, содержащее не номер регистра, а непосредственное (Indirect), также обозначаемое как literal, («буквальный») представление числа. Эта команда может быть смоделирована оператором вида:

Case 2 : reg[dest] = literal; break;

Недостатком такого формата является ограниченный размер поля literal. В приведенном примере он имеет размер 16 бит, что не позволяет загружать одной командой 32-разрядные регистры. Кроме того, поле literal не требуется для команд, использующих в качестве операндов регистры процессора. Однако в учебных целях, для сохранения единообразия представления команд, можно проиллюстрировать работу процессора на упрощенном примере.

Второй необходимый вариант команд – команды переходов. Без этих команд процессор будет иметь возможность выполнять только линейную последовательность операторов, не поддерживая ни условные операторы, ни циклы, ни вызовы подпрограмм. Для реализации практических примеров необходимо, чтобы кроме перехода к следующей команде, была доступна возможность перехода к заданному адресу.

Примеры имитации переходов приведены ниже:

Case 3 : pc = literal; break;

Case 4 : if (reg[op1] == reg[op2]) pc = literal; else pc = pc + 1; break;

В первом случае (тип команды указан равным 3) счетчик команд загружается значением, переданным в поле литерала. Это вариант безусловного перехода (jump). Во втором случае в pc либо загружается литерал, либо выполнение программы продолжается со следующего адреса. Это вариант условного перехода (т.е. перехода, выполняющегося при наступлении определенного условия). В примере показано условие, состоящее в равенстве значений регистров процессора с номерами op1 и op2.

Как правило, в процессорах используются разные варианты условий перехода. Обычно это переходы, выполняемые в зависимости от состояния однобитных переменных, устанавливаемых по результатам предварительно выполненных операций. Такие переменные называются флагами и входят в состав программной модели процессора (в рассматриваемом примере флаги не используются). Наиболее распространенные флаги:

– флаг нуля (Zero Flag), равен 1, если последний из вычисленных процессором результатов был равен 0;

– флаг переноса (Carry Flag), равен 1, если последний из вычисленных процессором результатов потребовал перенос в несуществующий старший разряд или заем из несуществующего старшего разряда.

При использовании флагов команды условного перехода выглядят следующим образом:

If (cf == 1) pc = literal; else pc = pc + 1;

If (cf == 0) pc = literal; else pc = pc + 1;

Рассмотренная модель имеет целый ряд упрощений:

– используются только целочисленные форматы данных;

– не учитывается количество тактов, затрачиваемое на выполнение команд;

– не моделируется работа системной шины и внешних устройств.

**Исходные данные**

|  |  |
| --- | --- |
| **Переменная** | **Описание регистров** |
| cmd | Переменная для обработки команд |
| pc | Счётчик команд |
| cmem | Память команд |
| dmem | Память данных |
| cmdtype | Тип команды |
| literal | Непосредственное значение |
| dest | Номер регистра для записи результата |
| Op1 | Номер регистра – первого операнда |
| Op2 | Номер регистра – второго операнда |

**Результат**

Выходные данные программы по разработке модели процессорного ядра программно-аппаратного комплекса представлены в виде скриншота окна консоли. Первая цифра – номер счётчика команд, вторая – результат работы алгоритма.

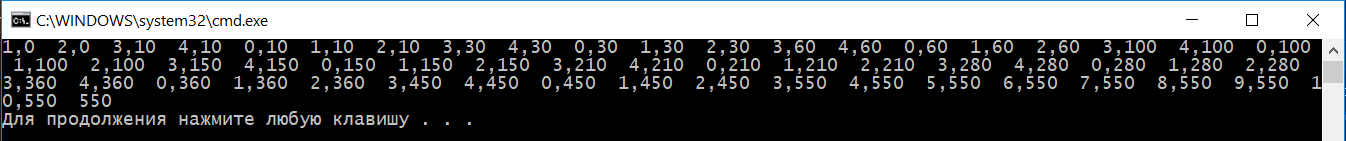


Рис. 6. Скриншот результатов работы модели.

**Выводы**

В ходе данной работы разработана программа, которая моделирует работу модели процессорного ядра программно-аппаратного комплекса и выдаёт результаты в консоль.

Разработка модели процессорного ядра программно-аппаратного комплекса выполнена на языке программирования C# с помощью IDE Microsoft Visual Studio.

**Приложение**

using System;

namespace Assembler\_Lab1

{

class Program

{

static void Main(string[] args)

{

int pc = 0;

int[] Reg = new int[16];

int[] cmem = new int[1024];

int[] dmem = { 10, 20, 30, 40, 50, 60, 70, 80, 90, 100, 110, 120, 130, 140, 150, 160 };

int cmd, cmdtype, literal, dest, op1, op2;

cmem[0] = 0x1000AA00;

cmem[1] = 0x20000015;

cmem[2] = 0x30000001;

cmem[3] = 0x40001553;

cmem[4] = 0x500000A5;

while (pc < 10)

{

cmd = cmem[pc];

cmdtype = (cmd >> 28) & 0x0F;

literal = (cmd >> 12) & 0xFFFF;

dest = (cmd >> 8) & 0x0F;

op1 = (cmd >> 4) & 0x0F;

op2 = cmd & 0x0F;

switch (cmdtype)

{

case 1: Reg[dest] = literal; pc++; break;

case 2: Reg[op1] = dmem[Reg[op2]]; pc++; break;

case 3: Reg[dest] = Reg[op1] + Reg[op2]; pc++; break;

case 4: Reg[dest] = Reg[op1] + literal; pc++; break;

case 5: if (Reg[op1] != Reg[op2]) { pc = 0; } else { pc++; } break;

default: pc++; break;

}

Console.Write(pc + "," + Reg[0] + " ");

}

Console.WriteLine(Reg[0]);

}

}

}