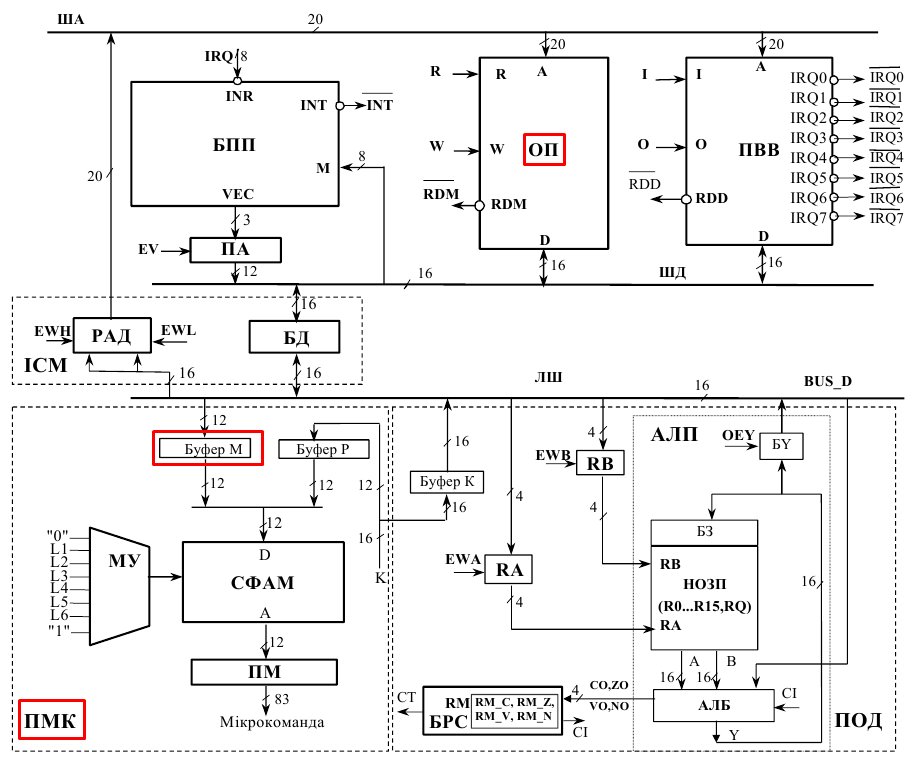
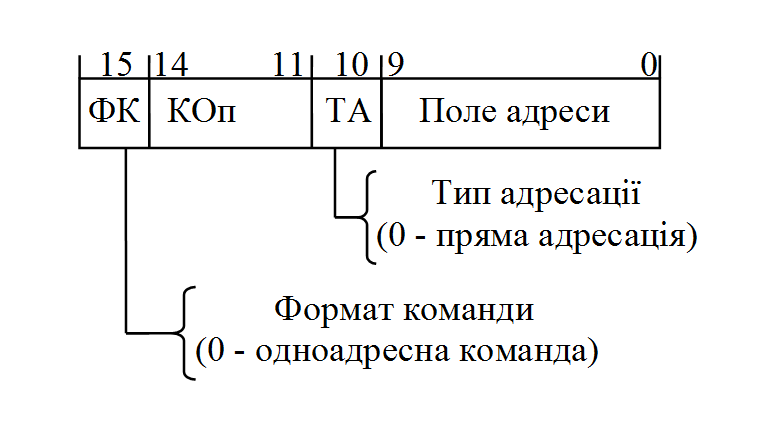
И так, начнём. В этой лабораторной работе нам нужно сделать что то наподобии методов в JAVA, которые мы сможем вызывать, забивая тот или иной адрес.

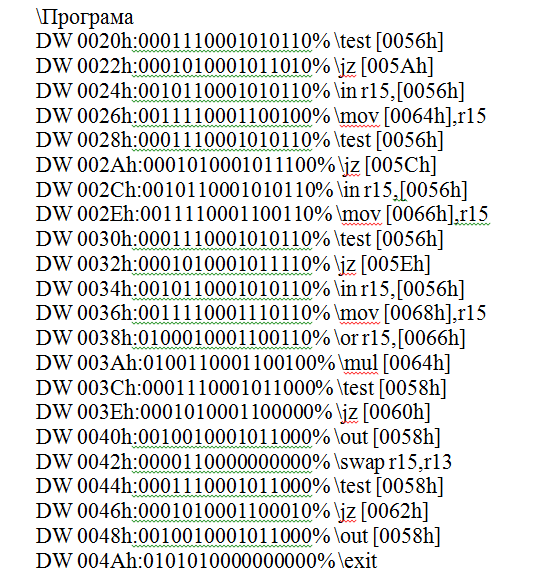
Что же отвечает за формировку команды?

  
Всем рулит ПМК, у него внутри есть память, в которую заносятся микрокоманды.  
Программа начинает работать с адреса 0h.

Но прежде чем вернуться к ПМК, мы рассмотрим структуру команды:

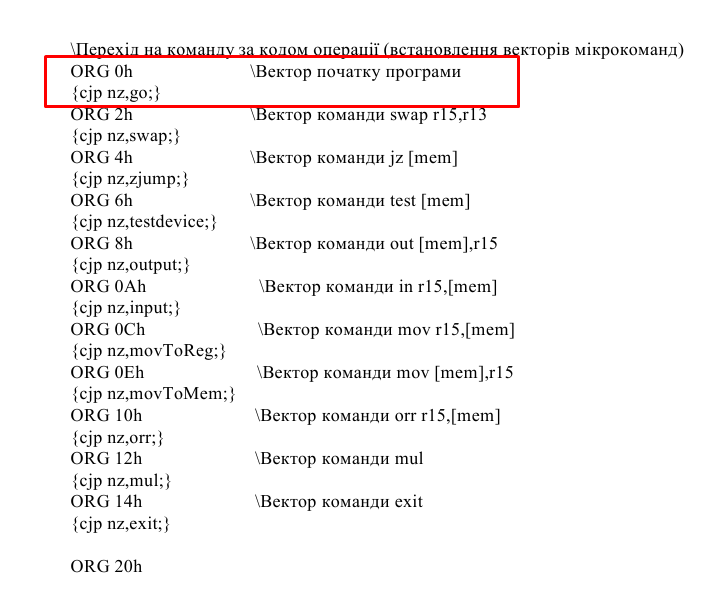


То бишь в память вы не будете вносить отдельно адреса команды, адреса операнда, вы запишете всё это в одну строку, на рисунке можно увидеть, какой бит за что отвечает.



В память вы вбиваете команды которые идут последовательно, на рисунке видно, что все адреса возрастают на 2, то есть все команды будут выполняться по-порядку.

Теперь вернемся к нашему ПМК, будьте внимательны, мы используем директиву ORG . Командой ORG в память ПМК заносятся куски исполняемого кода по определённому адресу.



То есть по нулевому адресу в ПМК будет переход на команду go

Но важно запомнить что 11 – 14 биты в вашем формате команд должен соответствовать адресу в ORG.

Ну например:

У вас есть такой формат: 0 0001 1 0000000101

0001 это адрес команды, на кототорый вы будете перескакивать

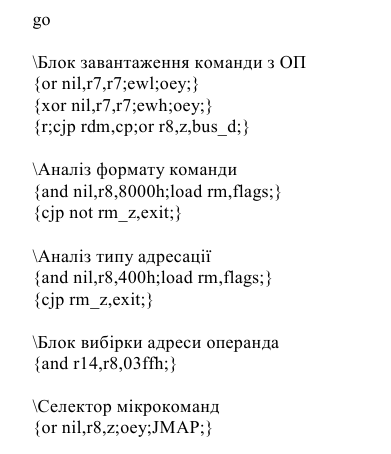
Ну например у вас команда mul

В ORG вы должны записать:

ORG 1h {cjp nz, mul}

И теперь если встретится адрес 0001, оно перепрыгнет на метку mul

Теперь рассмотрим команду go



В этих строках идет расшифровка вашего формата команды, а также её анализ: если формат команды не соответствует требуемому – происходит выход из программы (exit). Если же всё ОК, мы прыгаем по адресу в ПМК на команду.

Обращаем внимание на последнюю строчку. Команда JMAP осуществляет переход в памяти ПМК по адресу, который лежит в Буфере М.

Как туда попал адрес команды?

Он туда пишется в этой же строчке, командой or nil, r8, z значение из r8 (в котором сейчас значение типа 0001110001010110%) попадает на БY. Далее это значение с помощью сигнала OEY попадает на ЛШ. С ЛШ оно автоматически попадает в Буфер М.

Но как ПМК понимает где в этой строчке (0001110001010110%) адрес следующей микрокоманды?

Дело в том, что в Буфер М попадает не вся эта строка. В самом начале программы мы видим следующую строчку LINK M:z,z,z,z,z,z,z,14,13,12,11,z. В этой строчке мы говорим регистру М (читай Буферу М) какие биты (из 16 битов ЛШ) записывать в 12-битный Буфер М. Если детальнее, то: в самый младший бит Буфера М будет всегда писаться ноль (z значит 0), в следующий за ним будет писаться 11-й бит ЛШ (!!! обращаем внимание, что здесь биты ЛШ нумеруются начиная с ноля). Далее пишется 12, 13, 14 (14 – получается что предпоследний), а дальше нули.

Чтобы было легче понять, пронумерую биты ЛШ следующей табличкой, сразу на примере строки, которая упоминалась выше:

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 15 | **14** | **13** | **12** | **11** | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | **0** | **0** | **1** | **1** | 1 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 1 | 0 |

И вот что записывается в Буфер М:

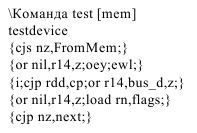
|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 11 | 10 | 9 | 8 | 7 | 6 | 5 | 4 | 3 | 2 | 1 | 0 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | **0** | **0** | **1** | **1** | 0 |

То есть теперь, с учётом самого младшего нолика, в ПМК будет искаться адрес 110, то есть 6h.

А вот что у нас записано в 6h:

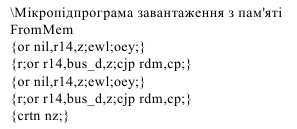


То есть переход. Переход на метку testdevice:



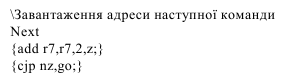
И в самой же первой строчке ещё один переход, на метку FromMem:

В r14 записаны младшие биты вашего формата команды, и по этому адресу оно в ОП находит операнды

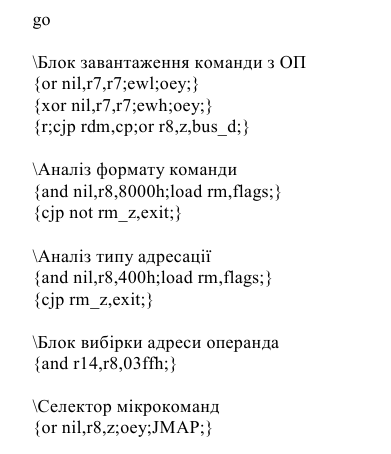


Здесь какие-то операции. А последняя строчка crtn nz означается возврат туда, откуда была вызвана эта подпрограмму (простыми словами – откуда вызвали эту метку), то есть возвращаемся в testdevice.

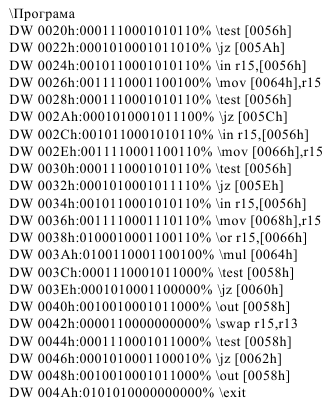
В testdevice в последней строчке снова переход. В этот раз на метку next:



Что делают эти строчки? Увеличивают значение регистра r7 на 2, что бы работать уже со следующей комадой . Далее переход на метку go, с которой мы начинали.



r7 используется для запоминания адреса текущей команды в ОП (основной памяти). Не путать ОП с памятью ПМК. Да именно там хранятся все наши строчки типа 0001110001010110%. И заносятся они туда командой DW:



И если присмотреться внимателнее, то в начале программы мы увидим, что r7 инициализируется со значением 0020h



И в подпрограмме go в самом начале мы берём строчку 0001110001010110% именно по такому адресу.

Далее в подпрограмме next мы увеличиваем значение r7 на 2, то есть теперь там будет 0022h. А значит, следующая строка - 0001010001011010%.

Процесс начинается сначала.

Таким образом, мы разбиваем всю программу на команды и последовательно их выполняем, работая при этом с памятью ПМК.



АВТОР: Андрей Михацкий.

*РЕДАКТОРЫ*: Виктор Иванов, Максим Морозов.