Делаем лабу 7 по ОПД

методичка для котят

1 Что от нас хотят

Задание:
Синтезировать цикл исполнения для выданных преподавателем команд. Разработать тестовые программы, которые проверяют каждую из синтезированных команд. Загрузить в микропрограммную память БЭВМ циклы исполнения синтезированных команд, загрузить в основную память БЭВМ тестовые программы. Проверить и отладить разработанные тестовые программы и микропрограммы.

Вариант: **1781**
MNEG M - изменение знака ячейки памяти, установить признаки N/Z/V/C Код операции - 9xxx
Тестовая программа должна начинаться с адреса $024B_{16}$

НО ЧТО ЭТО ЗНАЧИТ???

Мы все помним, что наша БЭВМ использует шины и вентили, чтобы пересылать данные между регистрами. Каждый регистр (почти) имеет вентиль, подающий данные В регистр, и вентиль, контролирующий чтение ИЗ регистра.
Каждая *команда* БЭВМ является набором инструкций, открывающих и закрывающих эти вентили - *микрокоманд (МК)*. Характерная черта микрокоманды - она выполняется ровно за один такт процессора
В нашей седьмой лабе нам нужно написать свою последовательность МК, выполняющих функцию в варианте.

2 Структура микрокоманды

Типы микрокоманд:

1. Операционная (ОМК) - выполняет операцию с данными
2. Управляющая (УМК) - команда ветвления

ОМК и УМК можно отличить по старшему, 39-ому биту.

Одинаковая струтура у них в младших 16-ти битах, остальное - отличается разительно. Здесь я отошлю вас в методичку, гляньте схему и запомните.

3 Выполнение лабы

Когда мы выполняем нашу лабу, мы внимательно смотрим на шпаргалку вентильных схем в презентации и картинку БЭВМ со всеми вентилями.

Советую обратиться к лабе Барсукова *(будто вы этого не сделали, проказники)*, а также к реализации команд БЭВМ в самой БЭВМ
Для этого в дуалмоде белой версии прописываем команду **mdecodeall**

```
PS C:\Users\Лев\Downloads> java -jar -Dmode=dual bcomp-ng.jar
Эмулятор Базовой ЭВМ. Версия v1.45.09 re9000611d74ef30fad84dc6ece6158cda
БЭВМ готова к работе.
Используйте ? или help для получения справки
mdecodeall
Адр    МК       Метка           Расшифровка
00 4000000000                   Halt
01 00A0009004   INFETCH         IP ? BR, AR
02 0104009420                   BR + 1 ? IP; MEM(AR) ? DR
03 0002009001                   DR ? CR
04 8109804002                   if CR(15) = 1 then GOTO CHKBR @ 09
```

Рис. 1: Знак вопроса заменяет в терминале символ передачи данных '→'

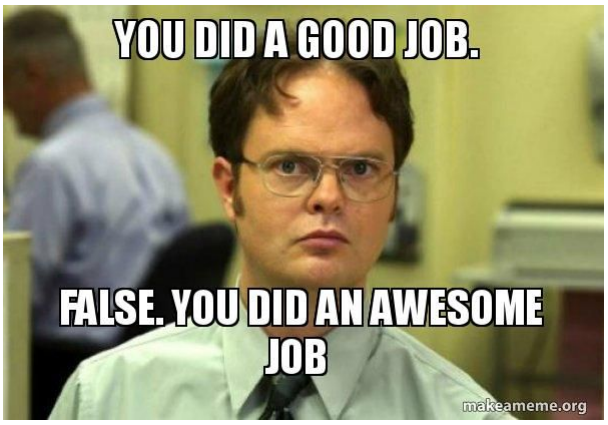
В поле МК вы обнаружите те самые микрокоманды, записанные в формате десяти 16-ричных чисел, перевести в двоичный код можно питоном или GPT.
Кстати о нем!
Если скормить модели схемы микрокоманд и семантику каждого бита, то она сможет на ура расшифровывать микрокоманды и предсказывать, что будет делать отдельная МК или даже их последовательность.

3.1 Полезные микрокоманды

- Переход на цикл прерывания - 80C4101040.
  Эта команда **обязательно** должна быть в конце вашей микропрограммы, иначе ничего работать не будет
- Сохранение DR в память (STORE) - 0200000000.
- Чтение в DR по адресу из AR - 0100000000

Остальные смотрим в БЭВМ или делаем сами

4 Ну написал, а дальше чё??



Теперь мы переводим все наши команды в формат из 10 циферок и открываем дуалмод
`java -jar -Dmode=dual bcomp-ng.jar`

Тут набираем:
ma
mw микрокоманда1
mw микрокоманда2
...
mw микрокомандаN
decodeall

Смотрим, правильно ли у нас все записалось: Если да - вперед тестировать!!

```
DF 8001101040                    GOTO INFETCH @ 01
E0 0001E09501   RESERVED         ~DR + 1 ? DR, N, Z, V, C
E1 0200000000                    DR ? MEM(AR)
E2 80C4101040                    GOTO INT @ C4
```

Рис. 2: Ещё раз заостряю внимание на последней команде - переходе на интерапшн

5 Тестирование

Вот здесь, дорогие мученики Е.Блохиной (м-да, стоит сменить формат обращения), начинается веселье. То что написано в методичке - вас не касается!
Разберем на примере адресной команды формата 9xxx
Требования методички:

1. Проверка результата с различными аргументами
2. Проверка признаков результата (регистра PS)
3. Запись флага пройденных тестов в память

В ГИЕНУ ОГНЕННУЮ
Что реально нужно сделать, так это проверить, как работает ваша команда с различными режимами адресации

На моем опыте могу сказать, что все типы адресации сделать не просят, достаточно четырех. Внизу будет пример тестирующей программы, тестирующей поведение проги в режимах:

- Прямая абсолютная 90xx
- Косвенная относительная 98xx
- Прямая относительная 9Exx
- Прямая загрузка 9Fxx

```
ORG 0x11
X1: WORD    0x1234
X2: WORD    0x1234


CHECK:  WORD    0xEDCC ; check value for our command - just to not forget

ORG 0x24B
START:
    CALL    TEST1
    CALL    TEST2
    CALL    TEST3
    CALL    TEST4
    HLT


TEST1:
    WORD    0x9011
    RET



TEST2:
    WORD    0x9801
    RET
ADDR_X2:    WORD    0x12


TEST3:
    WORD    0x9E01
    RET
X3: WORD    0x1234

TEST4:
    WORD    0x9F34
    RET
CHECK4: WORD    0xFFCC ; different argument (and answer) for direct load
```

Листинг 1: Обратите внимание как нашу команду использовать в ассемблере

А щас угадайте какая адресация работает через жопу...

6 Защита теории

Туть просто вопросы и иногда мои ответы

6.1 В какие регистры БЭВМ нельзя записывать?

Нельзя **записывать** в Клавишный (IR)
Нельзя **читать** из Адресного (AR)

6.2 Какие бывают МК, сходства и различия

Частично описал в пункте 2, частично поймете из презентации и методички. Кратко:

1. ОМК - операция
   (Чтение → АЛУ → Коммутатор → Запись → Память и ВУ)

2. УМК - переход
   (Чтение → АЛУ → Коммутатор (только [L/H]TO[L/H]) → Выбор бита коммутатора → Адрес перехода → Бит сравнения)

Одинаковыми будут младшие 16 бит (там чтение, АЛУ + половинка коммутатора)

6.3 Почему прямая загрузка работает некорректно

В цикле выборки операнда мы кладём в AR адрес аргумента с которым работаем, поэтому мы имеем возможность в цикле исполнения по этому адресу сохранить результат работы команды.

Но у прямой загрузки просто **нет этого цикла**, а значит в AR будет все еще лежать адрес самой команды из цикла выборки команды. Итог - результат вычисления **затрёт** собой команду в памяти, что все сломает, если мы попытаемся снова исполнить этот кусок кода.