Лабораторная работа 7
Проектирование ядра микропроцессора

1. **Цель лабораторной работы**:
овладеть знаниями и практическими навыками по проектированию сложных программноуправляемых вычислительных блоков, таких как ядро микропроцессора (CPU). Лабораторная работа также служит для овладения навыками программирования и отладки описания CPU на языке VHDL.

2. **Теоретические сведения**

Каждый микропроцессор включает в себя ядро и множество устройств, подключаемых к нему через его интерфейс, таких как внешние блоки памяти, устройства ввода-вывода, сопроцессоры, расширитель входа прерываний. Ядро микропроцессора описывается его архитектурой с точностью до временных задержек, особенностей элементной базы и т.п. В архитектуру ядра микропроцессора входят: система команд, конфигурация памяти, включая адресные пространства; системы адресации, прерываний, защиты памяти, поддержки надежности и тестирования; структура ядра, его интерфейс, основные рабочие режимы.

От выбора системы команд зависят основные параметры микропроцессора. Но оптимизация системы команд представляет сложный процесс разрешения ряда противоречий. Если выбирать формат команд, в котором команды закодированы компактным образом, то в результате получим программы, которые занимают минимум пространства в памяти программ. Но тогда усложняются схемы дешифрации команд и увеличивается задержка этих схем, что приводит к уменьшению быстродействия процессора.

Типичными подходами к уменьшению длины программ являются внедрение команд различной длины и системы сложных команд. В первом случае минимизация длины программных кодов напоминает метод кодирования Хафмана. Во втором случае одна сложная команда может заменить несколько простых команд. Правда, при этом число возможных команд ограничивается разрядностью поля кода операции и сложностью схемы управления. В обоих случаях существенно усложняются схемы дешифрации и выборки команд, затрудняется повышение быстродействия за счет распараллеливания вычислений.

Архитектура RISC – процессоров основана на том, что они имеют простой, пусть даже логически избыточный формат команд. Это обеспечивает несложную и следовательно, быструю дешифрацию команд и адресов, возможность выборки и дешифрации нескольких команд одновременно. В настоящее время существует тенденция к разработке систем команд таких, как у RISC – процессоров. При этом, если требуется сохранение программы в компактной форме, то после компиляции ее упаковывают специальным компрессором, а перед непосредственным исполнением – распаковывают аппаратным декомпрессором.

Все современные микропроцессоры исполняют команды в конвейерном режиме. При этом исполнение каждой команды проходит несколько последовательных стадий: выборка команды, ее дешифрация, выборка операндов, операция с операндами, запись результатов. Эти стадии выполняются на различных ступенях конвейера, так что несколько команд выполняются в процессоре одновременно, но на различных стадиях. Поток команд в таком командном конвейере обрывается на командах переходов. При этом конвейер обычно освобождается от недообработанных команд, а при выполнении команды по адресу перехода - заполняется снова. Часто такие простои конвейера минимизируют, применяя выполнение команд «задержанного» перехода.

3 **Пример описания CPU**

Рассмотрим пример проектирования простого 16- разрядного CPU на базе блоков AU, RAM и ICTR, спроектированных в предыдущих лабораторных работах. В CPU применяется непосредственная адресация (операнд непосредственно в команде) и косвенная адресация одного операнда в основной памяти, а также трехадресная адресация регистровой памяти. Адресное пространство процессора включает в себя 8 регистров FM и 8192 ячейки памяти RAM для хранения 16-разрядных данных и команд, а также 32 регистра периферийных устройств. CPU выполняет набор команд, представленный в таблице 8.

Таблица 8.

OP	W0							W1
	15 14 13	12 11	10	9 8	7 6	5 4 3	2 1 0	15…0
BRA	0 0 0	COND		DISP				
LJMP	0 0 1	ADDR						
CALL	0 1 0	ADDR						
LD	0 1 1	0 0	–	AQ		AB	–	
SD	0 1 1	0 1	–	–		AB	AD	
IN	0 1 1	1 0	APH	AQ		APL	–	
OUT	0 1 1	1 1	APH	–		APL	AD	
ALOP	1 0 0	ACOP		AQ		AB	AD	
LI	1 0 1	–	–	AQ		–	–	IDATA
RET	1 1 0	–	–	–		–	–	

По команде BRA выполняется условный переход относительно счетчика команд на смещение, определяемое непосредственной константой DISP. Так как эта константа представляет собой 10-разрядное число со знаком в доп. коде, то возможен переход на 512 ячеек памяти программ вперед и на столько же – назад. Семантика кода условия перехода COND приведена в табл. 9.

Таблица 9.

Мнемоника	COND	Функция от C,N,Z	Описание
NOP	0 0 0	0	Нет операции
JUMP	0 0 1	1	Безусловный переход
NEQ	0 1 0	Not Z	Результат ≠ 0
EQ	0 1 1	Z	Результат = 0
GE	1 0 0	not(C xor N)	Результат ≥ 0
LT	1 0 1	C xor N	Результат < 0
NCY	1 1 0	Not C	Перенос = 0
CY	1 1 1	C	Перенос = 1

По команде LJMP выполняется безусловный (длинный) переход по абсолютному 13-разрядному адресу ADDR. По такому же адресу выполняется вызов подпрограммы по команде CALL. При этом текущий адрес команды, увеличенный на 1, т.е. адрес возврата, должен сохраняться в регистре стека, т.е. в 7-м регистре FM (см. лабораторную работу 6). Противоположная ей команда RET выполняет возврат из подпрограммы и по ней адрес возврата из 7-го регистра переписывается в счетчик команд.

По команде LD данное из памяти по адресу, хранящемуся в регистре AB, загружается в регистр AQ. Т.е. здесь выполняется индексная адресация чтения из памяти. Аналогично по команде SD данное из регистра AD пересылается в память по адресу,

хранящемуся в регистре AB. Такая адресация позволяет выполнять доступ к дополнительной внешней памяти данных объемом до 56 Кслов.

Команды IN и OUT предназначены для ввода с периферийного устройства в регистр AQ и соответственно, вывода данного из регистра AD. При этом регистры периферийного устройства могут находиться в старших адресах адресного пространства процессора с младшими разрядами, задаваемыми 5-разрядным кодом AP, состоящим из поля старших разрядов APH и поля младших разрядов APL. Таким образом, процессор может иметь до 32 регистров периферийных устройств.

Команды ALOP выполняют арифметические и логические команды в AU (см. лабораторную работу 6). Их коды F и L образуют управляющее слово для AU, так что ACOP = F & L.

Команда LI выполняет загрузку в регистр AQ непосредственного операнда, т.е. константы IDATA, которая представлена во втором слове команды. Эта команда играет большую роль при задании адресов записи-чтения данных для команд LD и SD.

В системе команд имеется большое количество незадействованных комбинаций кодов команд. Эти комбинации можно будет использовать в будущем при необходимости расширения системы команд.

Разработанный в лабораторной работе 2 блок ICTR не соответствует данной архитектуре, так как обеспечивает приращение адреса до 4 единиц, в то время как требуется приращение и декремент до 512 единиц. Кроме того, блок должен выдавать адрес возврата при вызове подпрограмм и загружать в счетчик команд сохраненный адрес возврата при возврате из подпрограммы. Также он должен подавать на адресный вход RAM адрес данного при выполнении команд записи и чтения RAM. Структура модернизированного блока ICTR показана на рис. 14. Так как в ICTR добавлен вход I приращения адреса, вход ARETI загрузки адреса возврата, вход B адреса данного, то изменилась кодировка управляющего слова. Его семантика объясняется в табл. 10.

Табл. 10.

Команда	F	Действие
-	X 0 XX	Остановка
Не переход	X 1 0 0	CTR=CTR+1, A=CTR
LJMP, CALL	X 1 0 1	CTR=D, A=CTR
RET	X 1 1 0	CTR=ARETI, A=CTR
BRA	X 1 1 1	CTR=I, A=CTR
LD,SD	1 X XX	A=B

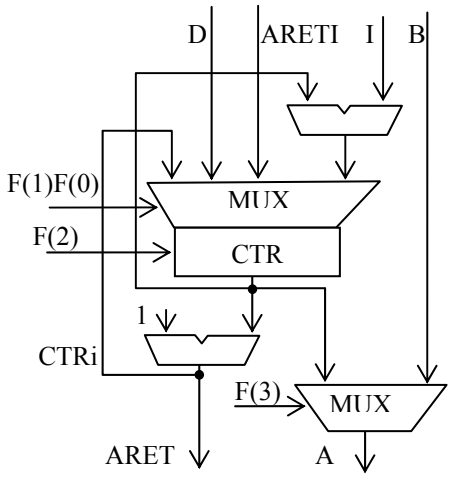

Рис. 14. Структура ICTR

Архитектура ICTR описывается так:

```
Library IEEE;
use IEEE.NUMERIC_BIT.all;
entity ICTR2 is
  port(CLK : in BIT;  -- синхровход
       RST : in BIT;  -- сброс
       D : in BIT_VECTOR(12 downto 0);  -- адрес перехода
       I :  in BIT_VECTOR(9 downto 0);   -- приращение адреса
       B : in BIT_VECTOR(12 downto 0);  -- адрес данного
```

```
        F : in BIT_VECTOR(3 downto 0);   -- функция
        ARETI: in BIT_VECTOR(12 downto 0);--адрес возврата
        A : out BIT_VECTOR(12 downto 0);-- выходной адрес
        ARET : out BIT_VECTOR(12 downto 0));-- адрес для сохранения
end ICTR2;
architecture BEH of ICTR2 is
  signal CTR,CTRi:SIGNED(12 downto 0);--счетчик адреса команды
begin
 ICTR:process(RST,CLK,CTR)
  begin
        CTRi<=CTR+1;       -- инкремент адреса
        if RST='1' then
              CTR <="0000000000000";  --сброс регистра счетчика
        elsif CLK='1' and CLK'event then
              case F(2 downto 0) is    -- мультиплексор и регистр адреса
                when "100"=> CTR<= CTRi;          -- увеличение на 1
                when "101"=>CTR<= SIGNED(D);    --загр. абс. адреса
                when "110"=>CTR<= SIGNED(ARETI);--загр. адреса возврата
                when "111"=>CTR<= CTR+SIGNED(I);-- с приращением
                when others => null;
              end case;
        end if;
  end process;
 MUX_A:A<=B when F(4)='1' else --выходной мультиплексор
        BIT_VECTOR(CTR);
     ARET<=BIT_VECTOR(CTRi);  -- адрес возврата из ПП
end BEH;
```

Здесь сигналы CTR, CTRi , соответствующие регистру – счетчику адреса и выходу схемы сложения с 1, объявлены как число со знаком. Поэтому к этим сигналам можно прибавлять 1 или число со знаком и такое суммирование будет отображено в соответствующие сумматоры чисел в дополнительном коде. Соответственно, входные сигналы, перед присваиванием или прибавлением к CTR переводятся в подтип SIGNED. И наоборот, выходные адреса при присваивании формируются как переход типа из SIGNED в BIT_VECTOR.

Структура процессорного ядра показана на рис. 15. Кроме ICTR, AU, RAM она содержит регистр команд IRG, командный блок управления COP и систему шин передачи данных и управления. Регистр команд IRG состоит из основной части IRG0 и части IRG1, сохраняющей непосредственный операнд. Шины адреса регистра ввода-вывода ADDRP и данных DI, DO с соответствующим мультиплексором MUXI образуют интерфейс ввода-вывода. При этом в периферийное устройство

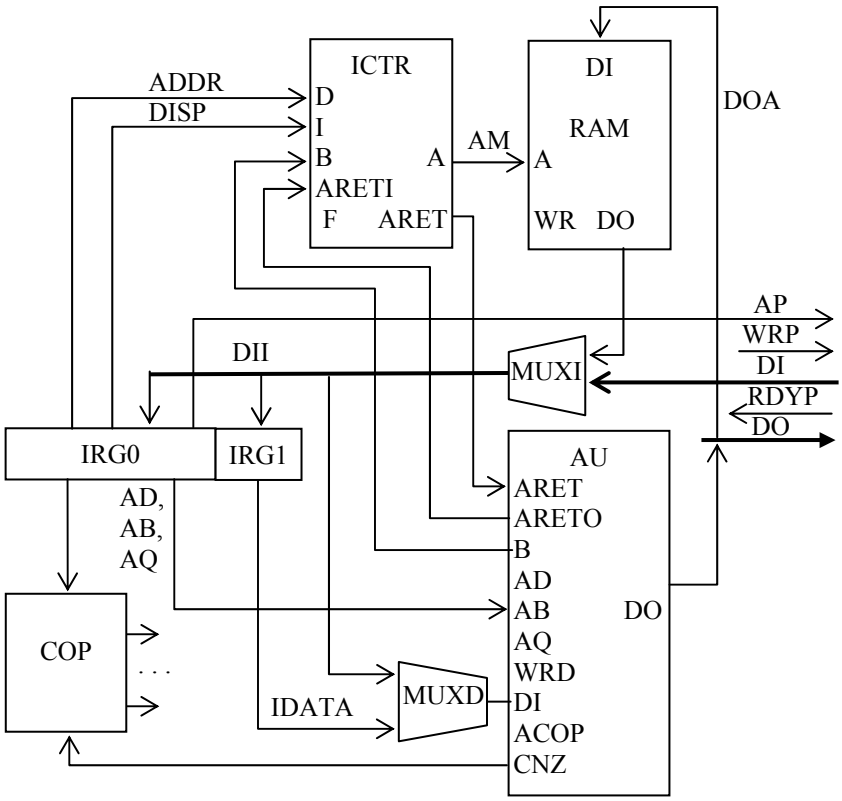

Рис. 15. Структура ядра микропроцессора

Интерфейс объекта AU имеет следующее описание.

```
entity AU is   port( CLK : in BIT;
        RST : in BIT;
        START : in BIT;      --начать операцию AU
        RD: in BIT;      -- чтение из FM на шину DO
        WRD : in BIT; -- запись с шины DI
        RET : in BIT; -- возврат из подпрограммы
        CALL: in BIT; -- вызов подпрограммы
        DI : in BIT_VECTOR(15 downto 0);    --вх. шина данных
        AB : in BIT_VECTOR(2 downto 0);      -- адрес регистра B
        AD : in BIT_VECTOR(2 downto 0);      -- адрес регистра D
```

```
        AQ : in BIT_VECTOR(2 downto 0);      -- адрес регистра Q
        ARET : in BIT_VECTOR(12 downto 0);    -- адрес возврата
        ACOP : in BIT_VECTOR(3 downto 0);     -- код операции AU
        RDY : out BIT;                        --готовность результата
        ARETO : out BIT_VECTOR(12 downto 0);-- адрес возврата
        DO : out BIT_VECTOR(15 downto 0); --вых. шина данных
        CNZ: out BIT_VECTOR(2 downto 0)); --вых. рег. состояний
end AU;
```

Описание архитектуры AU, в котором не приведены описания ее компонентов, выглядит следующим образом.

. . .

4 Испытательный стенд для AU

AU представляет собой сложный блок с внутренней памятью. Для его проверки предлагается испытательный стенд на основе простого устройства микропрограммного управления. Его описание архитектуры (кроме описания AU как компонента и сигналов – входов-выходов ALU) представлено ниже.

. . .

5 Вопросы по лабораторной работе.

Как выбирается перечень операций AU?

Какие источники операндов в AU?

Как повышают производительность AU?

Как в AU выполняют операции сдвига?

Как в процессорах организованы вызов процедуры и возврат из нее и какую роль в этом играет AU?

Какие признаки запоминают в регистре состояния AU?

Приведите пример процесса, описывающего регистр с мультиплексором на входе.

Как на VHDL запрограммировать ПЗУ микропрограмм?