4.7 Лабораторная работа 6
Проектирование арифметического устройства

4.7.1. **Цель лабораторной работы**:
овладеть знаниями и практическими навыками по проектированию вычислительных блоков, таких как арифметическое устройство (AU). Лабораторная работа также служит для овладения навыками программирования и отладки описания AU на языке VHDL.

4.7.2. **Теоретические сведения**

Блок AU является центральной частью практически любого процессора и предназначен для выполнения арифметических и логических операций с операндами, заданными в текущей команде. Перечень операций и операндов определен в системе команд данного процессора.

Перечень операций AU определяет производительность процессора, с одной стороны и сложность AU и дешифратора команд, с другой стороны. Минимальный перечень операций включает в себя операции, обеспечивающие алгоритмическую полноту системы команд. Это, как минимум, три логические операции, например, И, НЕ, установка в 0, арифметические операции сложения, вычитания, а также операции сдвига на один разряд вправо, выполняемые с целыми числами или числами с фиксированной запятой. Остальные арифметические и логические функции могут быть вычислены путем выполнения цепочек команд.

Для увеличения производительности процессора и сокращения длины программ в перечень команд вводятся операции вычисления функций, которые часто встречаются в реализуемых алгоритмах. Поэтому операции умножения и деления входят в систему команд большинства процессоров. Выбор перечня операций представляет собой поиск компромисса между удобством системы команд, объемом памяти, занимаемым програмой, быстродействием процессора и сложностью AU.

Операнды в AU могут поступать из регистровой памяти FM, оперативной памяти RAM или из специальных регистров, как например, из регистра – аккумулятора. Эти же блоки и регистры могуть служить приемниками результата AU. Признак переноса C участвует в операциях как особенный операнд.

Источники операндов определяются видами адресации системы команд. При использовании нескольких видов адресации входы AU должны содержать мультиплексоры операндов, которые подключают те или иные источники операндов. Такие мультиплексоры представляют собой дополнительные аппаратные затраты и вносят добавочную задержку в цикл выполнения команд.

В простейшем случае все операнды приходят из FM. Такая адресация операндов применяется в большинстве RISC – процессоров. Если FM и LSM реализованы в одном блоке AU, то большинство передач данных и арифметико-логических операций выполняется внутри блока, т.е. они являются локальными. При этом достигается минимальная длительность командного цикла.

Признаки результатов выполнения операций обычно запоминаются в специальном регистре, представляющем собой часть регистра состояния процессора. Кроме признака переноса C, знака результата N, нулевого результата Z, фиксируется также признак переполнения V, признаки исключительных состояний, например, деления на ноль, выход за допустимые границы адресного пространства и т.п. При операциях сдвига выдвигаемый разряд чаще всего запоминается в разряде признака переноса C.

Как правило, операция в AU выполняется за один такт, если операнды и результат хранятся в FM. Многие операции, как, например, операции умножения, деления в простейших AU, операции вычисления элементарных функций, операции с плавающей запятой, выполняются за несколько тактов. При этом AU должно работать под управлением специального автомата.

Для повышения производительности процессора уменьшают длительность тактового интервала путем вставки в AU регистров промежуточных результатов. При этом в AU организуется конвейерная обработка операндов, а в процессоре в целом - конвейерное исполнение команд. В конвейерном AU на соседних ступенях конвейера одновременно выполняются разные стадии различных операций. Такое AU также должно обеспечивать заполнение и очистку ступеней конвейера при выполнении программного перехода, сохранение и восстановление его состояния при вызове подпрограммы и возврате из нее. Поэтому конвейерное AU существенно сложнее, чем обычное AU.

4.7.3 **Пример описания AU**

Рассмотрим пример проектирования 16- разрядного AU, выполняющего набор операций, представленный в таблице 1 на базе трехкнального блока FM с восемью регистрами.

Таблица 4.7.

Операция	Мнемо-ника	Код операции ACOP	Код F управ-ления LSM	Бит переноса C0 или бит S15
Сложение	ADD	0000	00	0
Сложение с переносом	ADDC	0001	00	C
Сложение с инкрементом	ADDINC	0010	00	1
Вычитание с декрементом	SUBDEC	0100	01	0
Вычитание с переносом	SUBC	0101	01	C
Вычитание	SUB	0110	01	1
Логическое И	AND	1010	10	-
Логическое ИЛИ	OR	1110	11	-
Сдвиг вправо логический	SRL	1000	-	0
Сдвиг вправо с переносом	SRC	1001	-	C
Сдвиг вправо арифметический	SRA	1011	-	N
Умножение	MUL	1100	-	-

В состав AU входит блок LSM, выполняющий операции сложения и вычитания с переносом, логические И и ИЛИ. Операции сложения и вычитания образуются путем подачи 0 , 1 или флага переноса C на вход C0 блока LSM (см. табл.4.7). Для выполнения операции сдвига вправо необходимо подключить сдвигатель на один разряд. При этом для выполнения логического, арифметического сдвигов или сдвига с переносом в левый разряд результата вставляется 0, флаг C или флаг знака N. Операция умножения выполняется последовательно-параллельно в блоке умножения, разработанном в лабораторной работе 5.

Поскольку FM – трехадресный, то по одной команде выбирается 2 источника операндов и 1 приемник – результат. Как видно из табл.4.7, всего использовано 12 комбинаций кодов операции, т.е. в систему операций AU при необходимости можно добавить ещё 4 операции.

Операция умножения имеет 2 операнда и 2 шестнадцатиразрядных результата (старшая и младшая части произведения). Так как задан один адрес результата, то целесообразно 2 слова результата располагать в двух соседних регистрах, адреса которых

отличаются на 1. Для этого на адресном входе AQ блока FM следует вставить схему инкремента.

Все операции должны выдавать признаки результата C – бит переноса, Z – признак нулевого результата и бит знака N, которые должны запоминаться в регистре состояния. Таким образом, регистра 7. При выполнении вызова подпрограмм в области стека должно сохраняться состояние регистра состояния и адрес возврата из подпрограммы, а по команде возврата из подпрограммы этот адрес возвращается в ICTR. В RISC – процесорах при вызове подпрограмм адрес возврата сохраняется в одном из регистров блока FM, например, в последнем. Так как в учебном процессоре адресное пространство ограничено 13 разрядами, то целесообразно младшие разряды регистра 7 отвести под адрес возврата, т.е. биты C, Z и N будут сохраняться в 15, 14 и 13 разрядах этого регистра, соответственно.

Таким образом, для реализации всех функций AU необходимо внести изменения в блок FM, что приводит к изменению как его структуры, так и интерфейса. Структура блока FM показана на рис.4.11 , а структура блока AU в целом – на рис. 4.12.

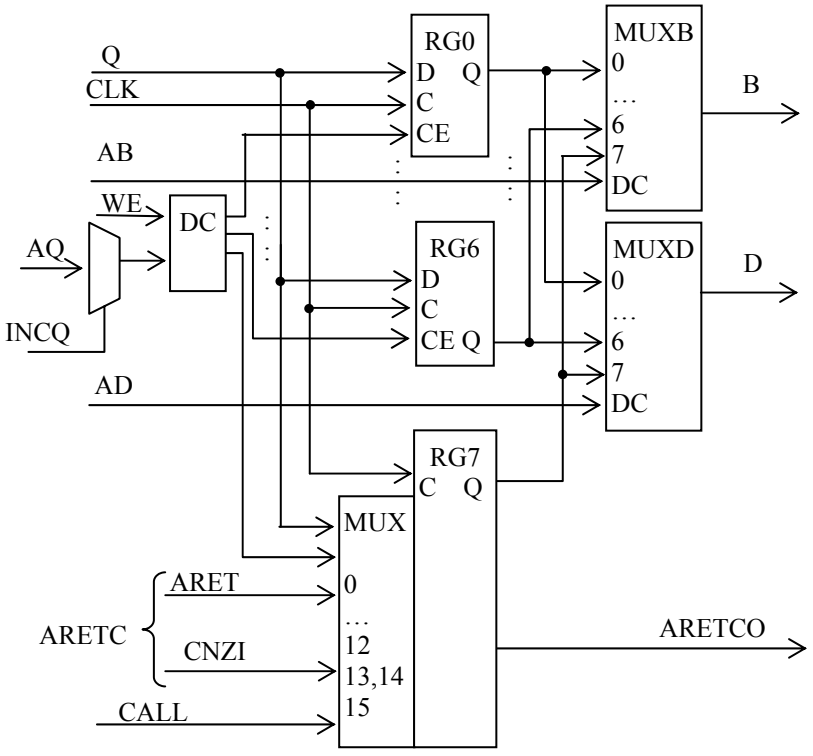

Рис.4.11. Структура модернизированного блока FM

Модернизированный блок FM имеет следующее описание.

```
Library IEEE;
use IEEE.NUMERIC_BIT.all;
entity FM2 is port(CLK:in BIT; -- синхровход
                   WR:in BIT;  -- сигнал записи
                   INCQ:in BIT;-- +1 к адресу Q
                   CALL:in BIT;-- запись адреса возврата и CNZ
                   AB:in BIT_VECTOR(2 downto 0);-- адрес канала B
                   AD:in BIT_VECTOR(2 downto 0);-- адрес канала D
                   AQ:in BIT_VECTOR(2 downto 0);-- адрес канала Q
                   ARETC:in BIT_VECTOR (15 downto 0);--адрес возврата и CNZ
```

```
            Q: in BIT_VECTOR (15 downto 0);-- данное канала Q
            B: out BIT_VECTOR(15 downto 0);-- данное канала B
            D: out BIT_VECTOR(15 downto 0);-- данное канала D
            ARETCO:out BIT_VECTOR (15 downto 0));--адрес возврата и CNZ
end FM2;
architecture BEH of FM2 is
      type MEM8X16 is array(0 to 7) of BIT_VECTOR(15 downto 0);
      signal RAM: MEM8x16;
begin
      FM8:process(CLK,AD,AB)              -- блок регистровой памяти ---
            variable addrq,addrd,addrb:natural;
      begin
            addrq:=TO_INTEGER(UNSIGNED(AQ));
            if INCQ='1' then
                  addrq:=addrq+1;
            end if;
            addrd:=TO_INTEGER(UNSIGNED(AD));
            addrb:=TO_INTEGER(UNSIGNED(AB));
            if CLK='1' and CLK'event then
                  if WR = '1' then
                        RAM(addrq)<= Q;  -- запись
                  end if;
                  if CALL = '1' then
                        RAM(7)<= ARETC;  -- запись адреса возврата
                  end if;
            end if;
            B<= RAM(addrb);-- чтение канала B
            D<= RAM(addrd);-- чтение канала D
            ARETCO<= RAM(7);
      end process;
end BEH;
```

Структура блока AU показана на рис. 4.12. В него кроме блока FM входят ранее разработанные блоки MPU и LSM. Мультиплексор MUXQ предназначен для подачи на выход DO блока и на вход записи Q входного данного, или результата LSM, или результата MPU, или считанного данного D, или его же, но сдвинутого вправо по командам SRL, SRC или SRA. Мультиплексор MUXC выбирает флаг переноса C или 0, или 1 для подачи на вход переноса LSM для выполнения команд сложения и вычитания.

Регистр состояния SR записывает в конце выполнения команд признак переноса C, признак отрицательного результата N и признак нулевого результата Z, причем, если это команда умножения, то состояние записывается с блока умножения, если выполняется команда возврата из подпрограммы RET – то состояние, запомненное в 7-м регистре FM, а иначе – с блока LSM.

На выход BO подается данное, прочитанное из FM по адресу AB. Это данное может использоваться как адрес при индексной адресации внешней памяти.

Управляющий автомат FSM_AU имеет 3 состояния: free -AU свободен, mpy - идет умножение, mpyl - конец умножения, в зависимости от которых и входных сигналов ACOP, START и окончания умножения rdym формирует необходимые управляющие сигналы. Его диаграмма состояний показана на рис.4.13.

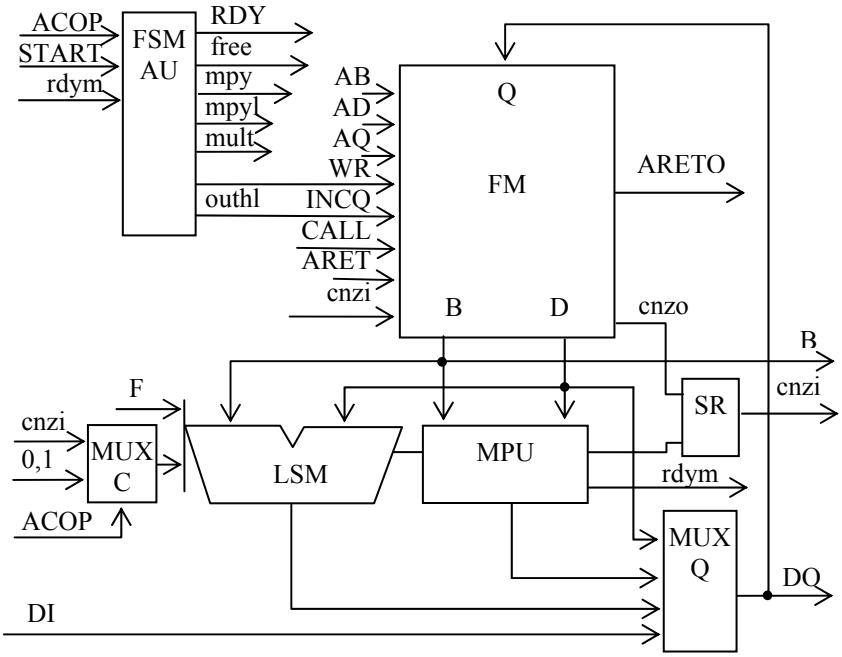

Рис.4.12. Структура блока AU

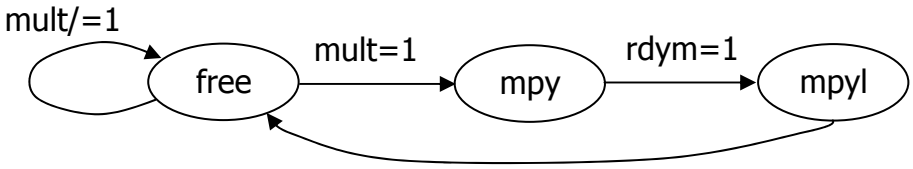

Рис.4.13. Диаграмма состояний FSM_AU

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
        BO : out BIT_VECTOR(15 downto 0); --вых. шина данного B
        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.

```
. . .
  type STAT_AU is (free,mpy,mpyl);-- состояния автомата
  signal st:STAT_AU;
  signal b,q,d,y,dp,aretc,aretco:bit_vector(15 downto 0);
  signal c0,c15,csh,zlsm,wr,mult,outhl:bit;
  signal rdym,zmpy,nmpy:bit;
  signal cnzr,cnzo,cnzi:bit_vector(2 downto 0);
begin
  U_FM: FM2 port map(CLK,        -- блок регистровой памяти
          WR=>wr,     INCQ=>outhl, CALL=>CALL,
          AB=>AB,    AD=>AD,      AQ=>AQ,
          ARETC=>aretc,
          Q=>q,        B=>b,        D=>d,
          ARETCO=>aretco);
  aretc<=cnzr&ARET;
  cnzo<=aretco(15 downto 13);
  ARETO<=aretco(12 downto 0);
  MUX_C:c0<='1' when ACOP(1 downto 0)="10"else  --мультиплексор C0
            cnzi(2) when ACOP(1 downto 0)="01"else '0';
  U_LSM:LSM port map(F=>ACOP(3 downto 2),  -- LSM
          A=>d,         B=>b,
          C0=>c0,      Y =>y,
          C15=>c15,   Z =>zlsm     );
  U_MPU:MPU port map(CLK,RST, -- блок умножения
          START=>mult,       OUTHL=>outhl,
          DA=>d,               DB=>b,
          RDY=>rdym,         Z=>zmpy,
          N=>nmpy,           DP=>dp);
  MUX_CI:csh<=cnzi(2) when ACOP(1 downto 0)="01" else --задвиг. разр.
            cnzi(1) when ACOP(1 downto 0)="11" else  '0';
   -----     мультиплексор результата ---------
  MUX_Q:q<=dp when st/=free else                  --результат умножения
          csh&d(15 downto 1) when ACOP="1000"   -- сдвиг вправо
                     or ACOP="1001" or ACOP="1011" else
         DI when WRD='1'   else                          --входное данное
         d when RD='1' else                             --данное из FM по адресу AD
         y;                                             --результат LSM
  SR:process(CLK,RST)           --регистр состояния с мультиплексором
  begin
          if RST='1' then
                  cnzi<="000";
          elsif CLK='1' and CLK'event then
                  if RET='1' then
                          cnzi<=cnzo;
                  elsif  st=mpyl then
                          cnzi<='0'&nmpy&zmpy;
                  elsif mult='0' then
                          cnzi<=c15&y(15)&zlsm;
```

```
                end if;
        end if;
  end process;
  mult<='1' when ACOP="1100" else '0';--дешифрация умножения
  FSM_AU:process(CLK,RST)              --автомат управления
  begin
        if RST='1' then
              st<=free;              -- регистр состояния автомата
        elsif CLK='1' and CLK'event then
              case st is
                    when free => if START='1'and  mult='1'then --свободен
                                       st<=mpy;
                                 end if;
                    when mpy=> if rdym='1' then -- идет умножение
                                       st<=mpyl ;
                                end if;
                    when mpyl=>  st<=free; --конец умножения
              end case;
        end if;
  end process;
--функции выходов автомата
  outhl<='1' when st=mpyl else '0';
  wr<='1' when WRD='1' or st=mpyl or (st=mpy and rdym='1')
        or (START='1' and mult='0')  else '0';
  RDY<='1' when st=mpyl or (WRD='0' and st/=mpy and mult='0')else'0';
  DO<=q; --выходное данное
   BO<=B;
  CNZ<=cnzi;  --выход регистра состояния
end BEH;
```

4.7.4 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.

```
. . .
  type MICROINST is record -- формат микрокоманды
        ACOP:bit_vector(3 downto 0);        -- код операции AU
        AQ,AD,AB:bit_vector(2 downto 0);   --адреса FM
        DI:BIT_VECTOR(15 downto 0);        -- входное данное
        START,WRD,RD:bit;                  -- биты управления
  end record;
  constant n: positive:=6;    --число микрокоманд
  type MICROPROGR is array(0 to n-1) of MICROINST;
  constant     mp:MICROPROGR:=(       -- ПЗУ тестирующей микропрограммы
  ("0000","001","000","000",X"4000",'0','1','0'),    --L 1,#4000h
  ("0000","010","000","000",X"cfff",   '0','1','0'),   --L 2, #-3001h
  ("0010","011","001","010",X"0000",'1','0','0'),    --ADDINC 3,1,2
  ("0101","011","001","010",X"0000",'1','0','0'),    --SUBC 3,1,2
  ("1100","100","011","010",X"0000",'1','0','0'),    --MUL 4,3,2
```

```
  ("0000","000","100","000",X"0000",'0','0','1'));  --S 4
  signal maddr:natural;
begin
  CLK<=not CLK after 5 ns; -- генератор синхросигнала
  RST<='1' ,'0' after 25 ns; -- генератор сигнала сброса
  CTM:process(CLK,RST) begin   -- счетчик микрокоманд
        if RST='1' then
              maddr<=0;
        elsif CLK='1' and CLK'event then
              if (RDY='1' and START='1') or WRD='1'or RD='1'  then
                    maddr<=(maddr+1) mod n; -- +1 к счетчику
              end if;
        end if;
  end process;
ROM_MP:(ACOP,AQ,AD,AB,DI,START,WRD,RD)<=mp(maddr);
UUT : AU port map (CLK,RST,         --тестируемое AU
        START => START,
        RD => RD,          WRD => WRD,
        RET => RET,        CALL => CALL,
        DI => DI,
        AB => AB, AD => AD, AQ => AQ,
        ARET => ARET,  ACOP => ACOP,
        RDY => RDY,  ARETO => ARETO,
        DO => DO, BO=>BO, CNZ => CNZ);
end TB_ARCHITECTURE;
```

Здесь типом MICROINST задан формат микрокоманды, подаваемой на входы тестируемого устройства. Поля микрокоманды можно добавлять при необходимости более глубокого тестирования. Константой – массивом mp задано содержимое ПЗУ микропрограммы, которое реализовано в операторе ROM_MP. Это содержимое представляет собой последовательность микрокоманд – входных воздействий для AU. Его можно изменять и дополнять в соответствии с алгоритмом тестирования. Длина микропрограммы задается константой n. В процессе CTM задан счетчик микрокоманд, который выполняет счет по модулю n.

Суть тестирования заключается в подаче данных в FM из полей микрокоманд, выполнение над ними операций в LSM и MPU, записи результатов в FM, чтении их из него и их проверке с контрольными результатами вычислений. Эта проверка может быть выполнена вручную при исследовании временных диаграмм в процессе моделирования. Также можно применить подход, использованный в предыдущей лабораторной работе, т.е. выполнить проверку результатов оператором **assert** в определенные моменты времени.

4.7.5 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?