Описание архитектуры учебной ЭВМ

Структура учебной ЭВМ

Учебная ЭВМ представляет собой 16-битную аккумуляторную архитектуру с общим адресным пространством, поддержкой базовой арифметики, загрузки/выгрузки данных, режимов адресации, стековых операций, прерываний и минимальной системы ввода-вывода.

Машина предназначена для учебных целей: демонстрации цикла «выборка-декодирование-исполнение», системных вызовов, прерываний, адресации памяти, обработки ошибок и низкоуровневой логики процессора.

ЭВМ состоит из процессора, оперативной памяти, регистров ввода и вывода, контроллера прерываний, кэш памяти и блока внешних устройств. Структура изображена на рисунке 1.

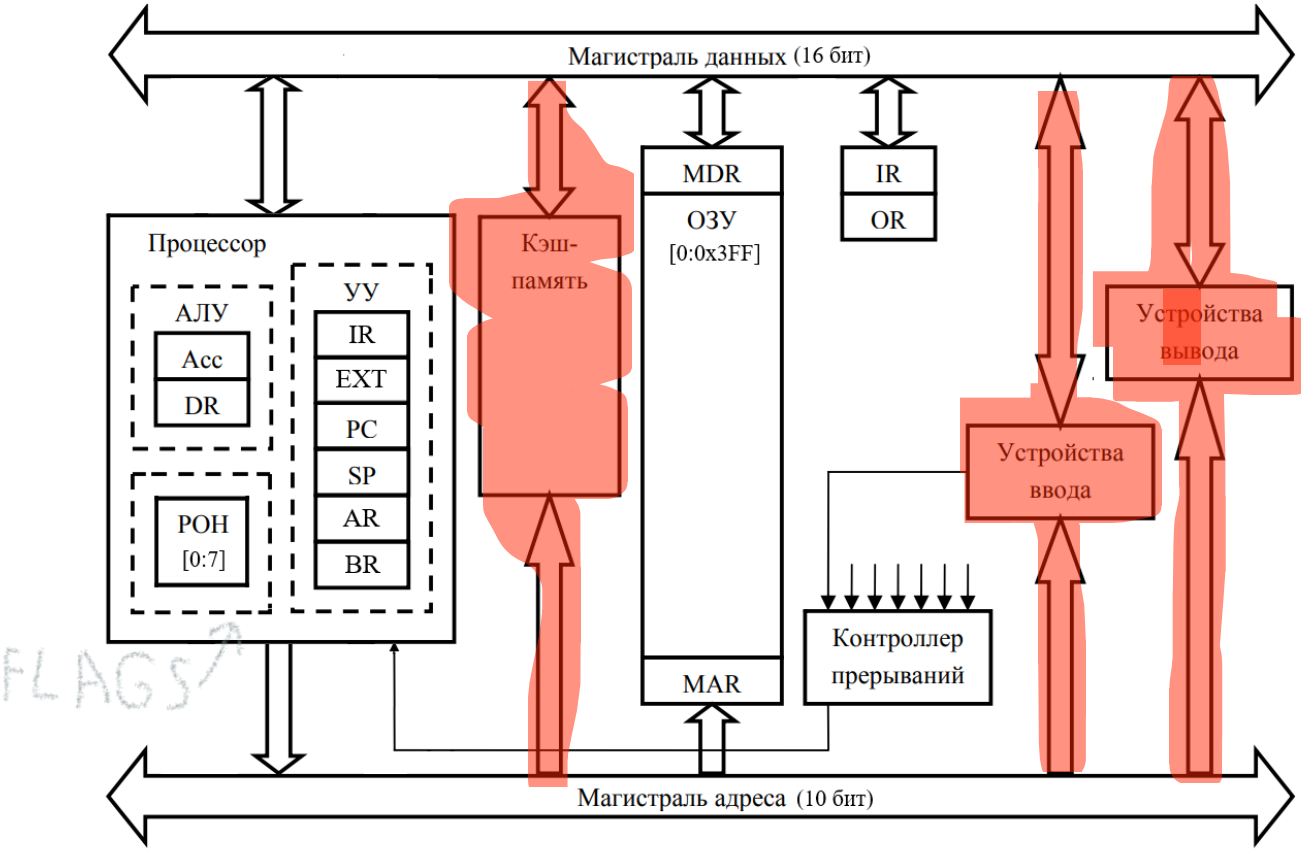

Рис. 1 Структура ЭВМ

Все устройства взаимодействую между собой через общие шины адреса и данных. Магистраль адреса имеет ширину в 10 бит, магистраль данных 16 бит.

Регистры

Архитектура ЭВМ основана на аккумуляторной модели и содержит фиксированный набор регистров, каждый из которых имеет размер 16 бит. Разрядность всех регистров едина, что упрощает реализацию и делает модель однородной.

Основные регистры процессора

1. ACC — Аккумулятор R/W
Главный рабочий регистр. Участвует во всех арифметико-логических операциях. Большинство инструкций предполагают использование аккумулятора как одного из операндов или как места для результата.

2. DR — Регистр данных (Data Register) R/W
Используется как временный регистр для операций загрузки/выгрузки, косвенной адресации и промежуточных вычислений.

3. PC — Счётчик команд (Program Counter) R/W
Содержит адрес текущей или следующей команды. Его значение обязано быть кратным 2 (так как инструкции занимают 2 или 4 байта).
При обработке переходов или прерываний может изменяться произвольно.

4. IR — Регистр команд (Instruction Register) R
Хранит 16-битную основную часть инструкции.

5. EXT – Регистр расширенных команд (Extendet) R
Если инструкция расширенная, хранит второй 16-битный блок.

5. SP — Указатель стека (Stack Pointer) R/W
Содержит адрес вершины стека.
Стек расположен в верхней части ОЗУ и растёт **вниз**, в сторону уменьшения адресов. Его значение всегда кратно 2.
При PUSH/SP-- и POP/SP++ происходит автоматическая корректировка указателя.

6. BR — Регистр базового адреса (Base Register) R/W
Используется в режиме относительной адресации как база при вычислении эффективного адреса:
EA = BR + offset.

7. AR — Регистр адреса (Address Register) R/W
Используется при косвенной адресации:
EA = MEM[AR]
При обращениях к памяти AR загружается эффективным адресом и участвует в последующем обращении через MAR/MDR.

8. MAR — Memory Address Register W
Адресный регистр подсистемы памяти. Содержит адрес ячейки памяти, к которой производится обращение.

9. MDR — Memory Data Register R/W
Регистр данных подсистемы памяти. Содержит данные, читаемые или записываемые в память через MAR.

10. IR / OR — регистры ввода/вывода (Input Register, Output Register) R и W
Минимальные порты для взаимодействия с внешним миром.

- **IR** — данные, записанные в него внешним устройством, могут быть прочитаны программой.
- **OR** — при записи в него данные считаются отправленными «наружу».

Используются как упрощённый механизм ввода-вывода без реализации контроллеров.

11. FLAGS — Регистр флагов R/W*
Содержит набор управляющих битов, используемых в условных переходах и обработке арифметики. Их список с описанием приведёт в таблице 1. Флаги автоматически изменяются АЛУ.

Таблица 1, список флагов

Бит	Имя	Назначение	~~Права~~
0	**Z** - Флаг нуля (Zero Flag)	Если результат операции равен 0, то Z = 1	~~R~~
1	**S** - Знаковый флаг (Sign Flag)	Старший бит результата равен 1 (отрицательное число в прямом коде)	~~R~~
2	**C** - Флаг переноса (Carry Flag)	Переполнение в арифметике без знака (перенос)	~~R~~
3	**O** - Флаг переполнения (Overflow Flag)	Арифметическое переполнение в операциях со знаком.	~~R~~
4	**I** - Разрешение прерываний (Interrupt Enable)	Разрешение аппаратных прерываний. Если I = 1, то прерывания разрешены, иначе все прерывания запрещены	~~R/W~~
5	**H** - Перенос из младшего байта (Half-carry / Auxiliary Carry)	Перенос из младшего байта (для будущих расширений)	~~R~~
6–15	—	Зарезервировано для расширения ISA	~~R~~

12. РОН — Регистры Общего Назначения R/W
Содержит 8 регистров R0–R7 общего назначения разрядностью 16 бит доступных для записи и чтения. Предназначены для хранения промежуточных данных.

Оперативная память

Объём ОЗУ и всего адресного пространства составляет 1024 байта (1 КБ). Используется побайтовая адресация. В итоге доступны адреса [0x03FF:0]. Все инструкции должны быть выравнены по четным адресам, поскольку каждой инструкции соответствует 2 или 4 байта в зависимости от формата.

Структура памяти

0x0000 – 0x001F — Таблица векторов прерываний
Каждое прерывание занимает 2 байта (адрес обработчика).
Всего: 16 × 2 = 32 байта.

0x0020 – 0x03FF — Программа, данные и стек.
Размещаются в одном сегменте — применяется модель фон Неймана.

Чтение/запись памяти

Процессор работает с памятью через MAR и MDR:

`MAR ← адрес`

`MDR ← MEM[MAR]` — при чтении

`MEM[MAR] ← MDR` — при записи

Это облегчает последующую реализацию микрокода.

Стек

Стек реализован в ОЗУ. При старте ЭВМ регистр SP (Stack Pointer) инициализируется значением 0x03FE, что является адресом последней команды в ОЗУ. Указатель стека всегда указывает на только на чётные адреса. Стек растёт в сторону уменьшения адресов.

Использование

Стек применяется для:

- сохранения PC и флагов при прерываниях;
- хранения временных данных и локальных переменных;
- вызова подпрограмм и возвратов (`CALL`, `RET`).

Защита и переполнение

В первой версии архитектуры защита не предусмотрена, поэтому переполнение или опустошение стека может привести к перезаписи данных о ошибках работы программы. Ответственность за тем, чтобы указатель стека

не попал в область данных программы или самой программы лежит на программисте.

Представление данных в ЭВМ

~~Все числа – целые. Представлены в двоичном виде в прямом коде, старший бит находится слева. Соответственно для 16 битного машинного слова, диапазон: от - 32767 до +32767 или от 11111111.11111111 до 01111111.11111111 в двоичном виде.~~

~~При переполнении чисел во время арифметических операций или операций сдвига, флаг O регистра FLAGS становиться активным (равным единице)~~

Все числа – целые. Они хранятся в системе в 16-битном двоичном представлении со знаком в формате дополнительного кода. Сложение и вычитание выполняются как обычное двоичное сложение по модулю 2^16, независимо от знака операндов. Знак итогового значения определяется автоматически в соответствии с дополнительным кодом.

Старший бит (бит 15) является битом знака:

- 0 — число положительное
- 1 — число отрицательное

Для 16-битного слова диапазон доступных целых чисел составляет:

- от –32768 (0x8000)
- до +32767 (0x7FFF)

То есть:

```
1000 0000 0000 0000b = -32768
0111 1111 1111 1111b = +32767
```

При выполнении арифметических операций АЛУ автоматически устанавливает флаги:

- Z (Zero) — результат равен нулю
- S (Sign) — старший бит результата равен 1
- C (Carry) — перенос из старшего разряда (используется в беззнаковой арифметике)
- O (Overflow) — произошло переполнение знаковой арифметики

Флаг переполнения O активируется, если:

- оба операнда имеют одинаковый знак
- знак результата отличается от знака операндов

При логических сдвигах вправо и влево:

- флаг C отражает вытесненный бит
- логический сдвиг вправо заполняет старший разряд нулём
- логический сдвиг влево заполняет младший разряд нулём

Формат инструкций

В архитектуре выделены 3 вида инструкций:

- 0 тип – (16 бит) инструкции без изменяемых параметров.
- 1 тип – (16 бит) короткие инструкции с параметрами
- 2 тип – (32 бита) длинные инструкции с параметрами

Тип инструкции – 0

Таблица 2, тип – 0

OPCODE	Не используется
[15:10]	[9:0]

OPCODE (6 бит)

Код операции — определяет тип выполняемой команды. Бит [15:15] всегда равен 0.

Тип инструкции – 1

Таблица 3, тип – 1

OPCODE	DATA
[15:10]	[9:0]

OPCODE (6 бит)

Код операции — определяет тип выполняемой команды. Бит [15:15] всегда равен 0.

DATA (10 бит)

Используется как данные/аргументы для инструкции, чаше всего обозначен 10 битный адрес. Подробнее смотрите в таблице 7, где указано значение для каждой отдельно взятой операции

Тип инструкции – 2

Таблица 4, длинная инструкция

Длинная инструкция		
OPCODE	**ADDRMODE**	**Не используется**
[15:10]	[9:7]	[6:0]

OPERAND
[31:16]

OPCODE (6 бит)
Код операции — определяет тип выполняемой команды. Бит [15:15] всегда равен 1.

ADDRMODE (3 бита)
Тип адресации. В таблице 5 в разделе адресации приведено сопоставление значений с типами адресации и их описание.

OPERAND (16 бит)
является аргументом инструкции. Он может означать:

- 10-битный адрес,
- 16-битную константу,
- смещение,
- аргумент для перехода,
- операнд для арифметики.

Декодер читает первое слово, определяет по биту [15:15] — нужна ли загрузка второго, если это бит равен единице, то инкрементирует PC ещё на 2 и загружает второе слово в регистр EXT

Типы адресации

Каждая команда содержит поле адресации размером 3 бита. Поле определяет способ интерпретации 7-битного аргумента и/или выбор одного из регистров общего назначения (R0–R7). Всего поддерживается 8 типов адресации, они приведены в таблице 4.

Таблица 5, типы адресации

Биты	Тип адресации	Обозначение в мнемокоде, x - адрес	Пример обозначения в мнемокоде
000	Прямая	x	ADD 0x22
001	Регистровая (РОН)	Rx	ADD R3
010	Непосредственная	#x	ADD #34
011	Косвенная через память	@x	ADD @0x22
100	Относительная от BR	[x]	ADD [2]
101	Регистрово-косвенная	@Rx	ADD @R3
110	Индексная с постинкрементом	@Rx+	ADD @R3+
111	Индексная с прединкрементом	-@Rx	ADD -@R3

Прямая
Объект находится непосредственно в памяти по адресу. То есть команда из примера ADD 0x22 будет брать в качестве второго операнда значение ячейки 0x22 из ОЗУ.

Регистровая
Объект находится в РОН с номером равным аргументу. То есть команда из примера ADD R3 будет брать в качестве второго операнда значение из 3 регистра РОН.

Непосредственная
В качестве операнда или значения подставляется непосредственно сам аргумент. То есть команда из примера ADD #34 будет брать в качестве второго операнда непосредственно значение 34.

Косвенная через память
Объект находится в памяти по адресу, который записан в ячейке памяти по адресу исходного аргумента. То есть при выполнении команды из примера ADD @0x22 предварительно будет прочитано значение ОЗУ в ячейке 0x22, а затем по этому значению будет браться значение из ОЗУ в качестве второго операнда.

Относительная от BR
Прямая адресация, где адрес объекта будет вычисляться как сумма аргумента и значения регистра BR. То есть команда из примера ADD [2] будет брать в качестве второго операнда значение ОЗУ в ячейке с адресом BR + 2.

Регистрово-косвенная
Объект находится в памяти по адресу, который записан в РОН по номеру исходного аргумента. То есть при выполнении команды из примера ADD значение второго аргумента будет браться из ОЗУ с адресом, который находится в регистре РОН R3.

Индексная с постинкрементом
Аналогично регистрово-косвенной, но после выполнения инструкции будет совершён инкремент адреса в регистре РОН с номером равным аргументу. То есть команда из примера @R3+ будет брать в качестве второго операнда значение из ОЗУ по адресу из регистра R3, а после исполнение инструкции увеличит значение R3 на размер инструкции

Индексная с прединкрементом
Аналогично регистрово-косвенной, но перед выполнением инструкции будет совершён декремент адреса в регистре РОН с номером равным аргументу. То есть команда из примера -@R3 сначала уменьшит R3 на размер инструкции,

а затем возьмёт в качестве второго операнда значение из ОЗУ по адресу из регистра R3

Модель прерываний

Процессор поддерживает **16 аппаратных прерываний** с фиксированной таблицей векторов, размещённой в общем адресном пространстве ОЗУ.

Модель прерываний предназначена для предоставления простой, понятной и учебной реализации механизма обработки событий, не усложняя устройство процессора.

Таблица векторов прерываний

Таблица векторов располагается в младшей области RAM, начиная с адреса 0x0000. Каждый вектор состоит из одного 16-битного слова — адреса обработчика прерывания. Итого на первых 32 байтах адресного пространства расположена таблица векторов прерываний.

Все прерывания имеют приоритет по номеру:
INT0 — самый высокий**,**
INT15 — самый низкий**.**

Таблица 6, вектора прерываний

Адрес	Название	Функция
0x0000	INT0	Старт/Перезагрузка ЭВМ
0x0002	INT1	
0x0004	INT2	
0x0006	INT3	
0x0008	INT4	
0x000A	INT5	
0x000C	INT6	
0x000E	INT7	
0x0010	INT8	
0x0012	INT9	
0x0014	INT10	
0x0016	INT11	
0x0018	INT12	
0x001A	INT13	
0x001C	INT14	
0x001E	INT15	

Источники прерываний

В архитектуре предусмотрены:

1. **Аппаратные прерывания** (например, таймер, внешние линии, I/O).

2. **Программное прерывание** (если введёшь инструкцию INT n — по желанию).

Условия входа в обработчик
Прерывание может быть выполнено только если:

1. Флаг FLAGS.I = 1 (прерывания разрешены)
2. CPU завершил выполнение текущей инструкции
3. Процессор не находится в режиме обработки прерывания с приоритетом выше текущего

Последовательность обработки прерывания
При возникновении прерывания и условия входа в обработчик удовлетворены, то процессор выполняет следующие шаги:

1. В стек сохраняются:
   - PC (адрес следующей инструкции)
   - FLAGS
2. Вычисляется адрес вектора
   `VectorAddress = InterruptNumber × 2`
3. Адрес вектора загружается в PC и происходит чтение следующей команды с нового адреса

Возврат из прерывания
Обработчик должен завершаться специальной инструкцией IRET, которая:

1. Восстанавливает FLAGS:
2. Восстанавливает PC:

IRET гарантирует корректное возвращение к основной программе.

Программные прерывания
Предусмотрена команда для программного вызова прерываний INT, где первые три младших бита инструкции – номер прерывания. Их поведение полностью совпадает с поведением аппаратных.

Система команд

В таблице 6 приведёт полный список команд. В ней используются следующие обозначения:

- MEM[<адрес>] – обращение к ячейки памяти с указанным адресом
- R(<номер>) – регистр общего назначения с указанным номером (от 0 до 7)
- DATA – непосредственный аргумент команды. 10 младших битов короткой инструкции 1 типа.

- operand – значение второго аргумента взятого из памяти или из регистра с учётом адресации для инструкций 2 типа
- = – операция присваивания
- == – проверка равенства
- В случае, если то или иное значение присваивается регистру PC то, всегда применяется маска 0x3FE, так как адрес инструкции всегда кратен двум

Таблица 7, полный список команд

Мнемокод	Код инструкции	Описание	Действие	Тип
Операции управления				
NOP	0x00	Пустая операция	Нет	0
EI	0x01	Разрешение прерываний	FLAGS.I = 1	0
DI	0x02	Запрет прерываний	FLAGS.I = 0	0
HLT	0x03	Остановка процессора	Нет	0
Арифметические и логические операции				
ADD	0x20	Сложение	ACC = ACC + operand	2
SUB	0x21	Вычитание	ACC = ACC – operand	2
CMP	0x22	Сравнение	ACC - operand	2
MUL	0x23	Умножение	ACC = ACC * operand	2
DIV	0x24	Деление	ACC = ACC / operand	2
MOD	0x25	Остаток от деления	ACC = ACC % operand	2
INC	0x0A	Инкремент	ACC = ACC + 1	0
DEC	0x0B	Декремент	ACC = ACC - 1	0
AND	0x26	Побитовое И	ACC = ACC & operand	2
OR	0x27	Побитовое ИЛИ	ACC = ACC | operand	2
XOR	0x28	Побитовое XOR	ACC = ACC ^ operand	2
NOT	0x0F	Побитовое НЕ	ACC = !ACC	0
SWL	0x10	Побитовый сдвиг влево	ACC = ACC << 1	0
SWR	0x11	Побитовый сдвиг вправо	ACC = ACC >> 1	0
Прерывания и стек				
PUSH	0x12	Поместить в стек	SP -= 2; MEM[SP] = ACC	0
POP	0x13	Извлечь из стека	ACC = MEM[SP]; SP += 2	0

PUSHF	0x14	Поместить в стек флаговый регистр	SP -= 2; MEM[SP] = FLAGS	0
POPF	0x15	Извлечь из стека флаговый регистр	FLAGS = MEM[SP]; SP += 2	0
CALL	0x16	Вызов подпрограммы	SP -= 2; MEM[SP] = PC PC = DATA[9:0]	1
RET	0x17	Возврат из подпрограммы	PC = MEM[SP]; SP += 2	0
INT	0x18	Программное прерывание	SP -= 2; MEM[SP] = FLAGS SP -= 2; MEM[SP] = PC PC = DATA[3:0]*2	1
IRET	0x19	Возврат из прерывания	PC = MEM[SP]; SP += 2; FLAGS = MEM[SP]; SP += 2	0
Ввод, вывод				
IN	0x1A	Ввод	ACC = IR	0
OUT	0x1B	Вывод	OR = ACC	0
Чтение и запись				
MOV	0x1C	Пересылка между РОН	R(DATA[5:3]) = R(DATA[2:0])	1
WR	0x29	Запись	R(DATA[2:0]) = ACC или MEM(DATA[9:0])) = ACC в зависимости от адресации	2
RD	0x2A	Чтение	ACC = VAL	2
WRBR	0x1F	Запись в регистр BR	BR = ACC	0
WRSP	0x04	Запись в регистр SP	SP = ACC	0
Переходы				
JMP	0x05	Переход без условий	PC = DATA[9:0]	1
JZ	0x06	Переход, если 0	IF FLAGS.Z == 1 PC = DATA[9:0]	1
JNZ	0x07	Переход, если не 0	IF FLAGS.Z == 0 PC = DATA[9:0]	1
JS	0x08	Переход, если отрицательно	IF FLAGS.S == 1 PC = DATA[9:0]	1
JNS	0x09	Переход, если неотрицательно	IF FLAGS.S == 0 PC = DATA[9:0]	1
JO	0x0C	Переход, если переполнение	IF FLAGS.O == 1 PC = DATA[9:0]	1
JNO	0x0D	Переход, если нет переполнение	IF FLAGS.O == 0 PC = DATA[9:0]	1
JNRZ	0x0E	Цикл		