Джон фон Нейман (1903-1957)

Архитектура Джона фон Неймана

– Кодирование двоичным кодом.

Основное же преимущество двоичной системы по сравнению с десятичной состоит в большей простоте и быстродействии, с которыми могут выполняться элементарные операции... Дополнительное замечание, которое заслуживает упоминания, состоит в том, что основная часть машины по своему характеру является не арифметической, а логической. Новая логика, будучи системой типа «да — нет», в основном двоична. Поэтому двоичное построение арифметических устройств существенно содействует построению более однородной машины, которая может быть лучше скомпонована и более эффективна» [18, стр. 28].

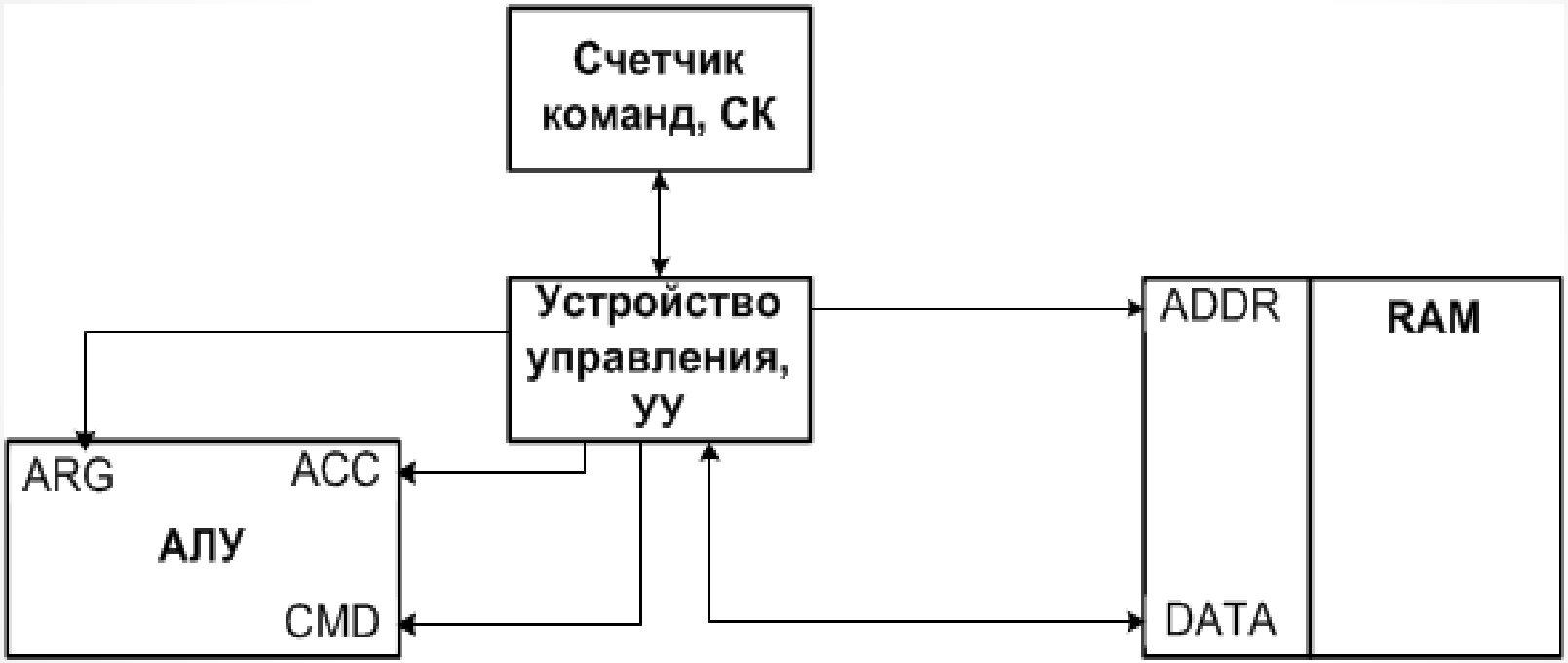
Переход на двоичную логику позволил использовать хорошо разработанный к тому моменту аппарат алгебры логики для анализа и синтеза узлов вычислительных машин.

Архитектура Джона фон Неймана

– Идея «хранимых программ»

1. Команды, так же как и числа, с которыми оперирует компьютер, записываются в двоичном коде.
2. Программа может храниться в том же запоминающем устройства, что и промежуточные результаты вычислений, константы и другие числа;
3. Появляется возможность перехода в процессе вычислений на тот или иной участок программы в зависимости от результатов вычислений, условных переходов.
4. Числовая форма записи программы позволяет производить операции над величинами, которыми закодированы команды программы - самомодификация кода

Архитектура фон Неймана



Сложим программно два числа и проанализируем результат

Область памяти	Адрес памяти	Содержимое ячейки	Пояснение
Команды	0x000	mov A, @0x100	Записать в А значение, хранящееся в ячейке 0x100 (это не то же самое, что записать значение 0x100!)
	0x001	add A, @0x101	Добавить к А значение из ячейки 0x101
	0x002	jz A, @0x004	Если в результате сложения получился 0, перейти к ячейке 0x010
	0x003	add A, 0x10	Добавить 0x10 к аккумулятору (на этот раз добавляется само значение 0x10, а не содержимое ячейки 0x10!)
	0x004	mov @0x102, A	Записать в ячейку с адресом 102 содержимое аккумулятора
	0x005	<завершение программы>	
	0x006		
	0x007		
	…		
	0x0FF		
Данные	0x100	0xFF	
	0x101	0x01	
	0x102		
	0x103		
	0x104		
	0x105		
	…		
	0x1FF		

Запуск процессора и выполнение первой команды

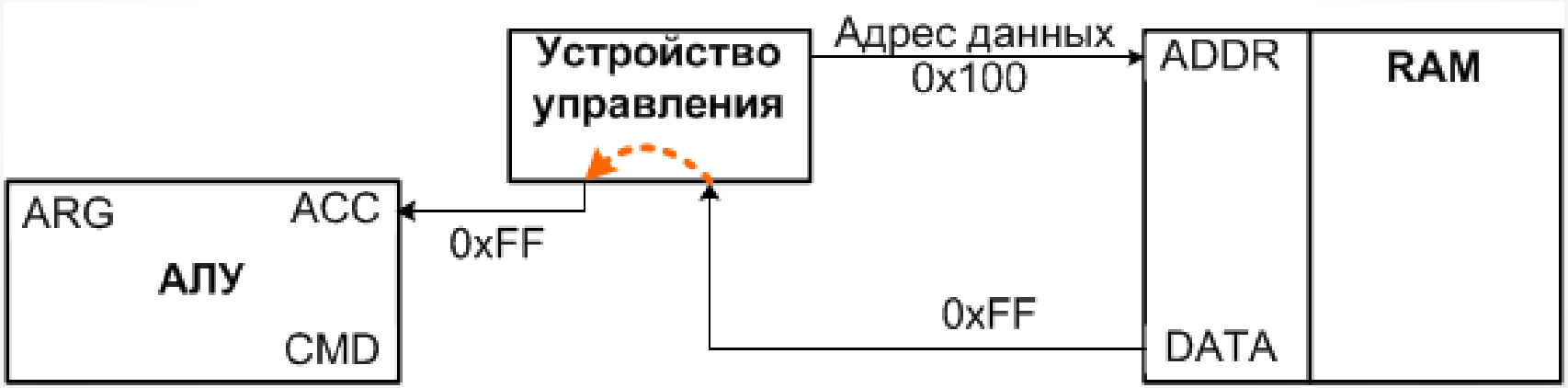
- СК изначально указывает на нулевой адрес памяти
- УУ пока «не знает», что ему нужно делать, но «знает» что по текущему адресу счетчика команд хранится инструкция (или команда), которую надо будет исполнить
- УУ читает команду по нулевому адресу памяти. По нулевому адресу лежит команда:

```
mov A, @0x100
```

- УУ теперь «знает», что ему сейчас делать – надо взять данные из ячейки с адресом 0x100 и скопировать их в аккумулятор

Как УУ копирует данные из заданной ячейки в аккумулятор

```
mov A, @0x100
```



Выполнение следующей команды

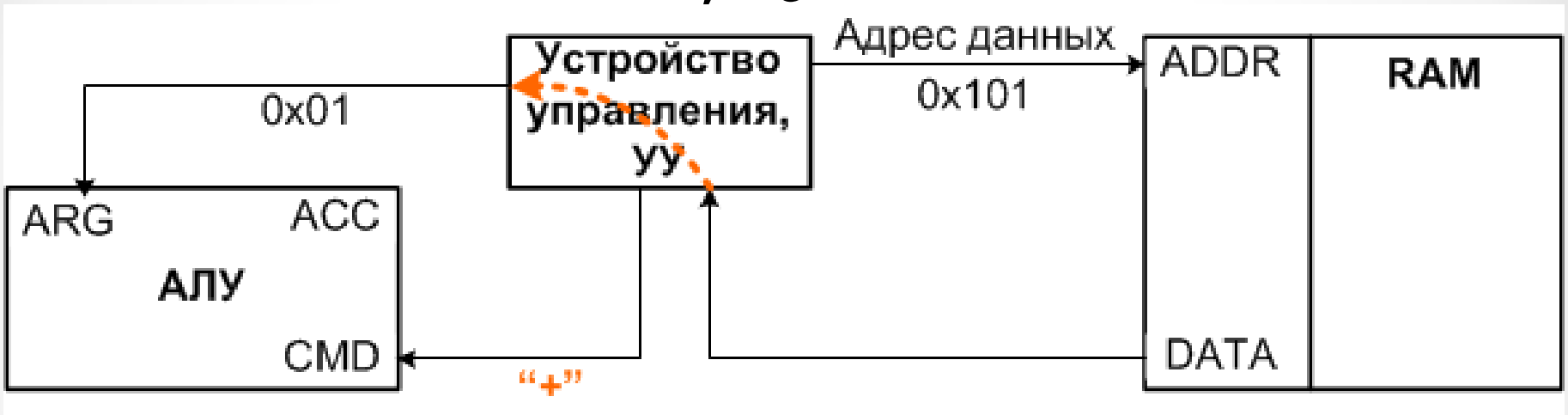
- После выполнения первой команды счетчик команд инкрементируется, теперь он указывает на ячейку с адресом 0x001. В этой ячейке лежит команда, которая требует сложить с аккумулятором данные из ячейки 0x101:

```
add A, @0x101
```

- УУ как и раньше узнает об этом, когда извлечет команду из памяти и внимательно на нее «посмотрит»

Как УУ складывает данные из заданной ячейки с аккумулятором

`add A, @0x101`



В результате АЛУ выполняет команду

`A += 0x01`

Предположим, что АЛУ восьмиразрядное:

$$A = 0xFF + 0x01 \bmod 2^8 =$$

$$= 0x100 \bmod 2^8 = 0$$

Выполнение команды условного перехода

- После выполнения команды сложения СК=0x002. По этому адресу лежит команда

`jz A, @0x004`

- Как мы помним, в аккумуляторе после сложения у нас лежит 0. Поэтому нужно сделать переход по адресу 0x004. Для этого на следующем шаге в счетчике команд должно оказаться не текущее значение плюс один (0x003), а значение 0x004.
- Собственно, УУ просто записывает 0x004 в СК.
- По адресу 0x004 лежит команда
- `mov @0x102, A`

Еще раз посмотрим на код. Что запишется по адресу 0x102?

Область памяти	Адрес памяти	Содержимое ячейки	Пояснение
Команды	0x000	`mov A, @0x100`	Записать в А значение, хранящееся в ячейке 0x100 (это не то же самое, что записать значение 0x100!)
	0x001	`add A, @0x101`	Добавить к А значение из ячейки 0x101
	0x002	`jz A, @0x004`	Если в результате сложения получился 0, перейти к ячейке 0x010
	0x003	`add A, 0x10`	Добавить 0x10 к аккумулятору (на этот раз добавляется само значение 0x10, а не содержимое ячейки 0x10!)
	0x004	`mov @0x102, A`	Записать в ячейку с адресом 102 содержимое аккумулятора
	0x005	`<завершение программы>`	
	0x006		
	0x007		
	…		
	0x0FF		
Данные	0x100	0xFF	
	0x101	0x01	
	0x102		
	0x103		
	0x104		
	0x105		
	…		
	0x1FF		

Самостоятельная проработка

- Как будет работать команда **`mov @0x102, A`** ?
- Как будет работать команда безусловного перехода **`jmp @0x020`** ?
- Все-таки, как УУ различает команды между собой? Чем отличаются ,например, от ?
- Как АЛУ различает действия между собой? Как умножить два числа или выполнить побитовую операцию на АЛУ?
- Как память с произвольным доступом отличает операции записи и чтения?

Эффективность архитектуры фон Неймана

- Еще раз подчеркнем, что команды и данные в архитектуре фон Неймана хранятся физически в одном и том же модуле памяти.
- В 1940-х годах это была прорывная идея, т.к. в результате использования электронной памяти вместо перфолент скорость работы ЭВМ значительно увеличилась и получилось легко организовать условный переход (см. команду jz).

Компьютер	Тип элементной базы	Представление чисел	Ввод программ	Наличие условных переходов	Разделение на команды и данные	Быстродействие
Z3, Германия, 1941	Электро-механический	двоичное	перфолента	нет	да	1 оп за 0.8 сек
МАРК-1, Гарвардский университет и IBM, 1944	Электро-механический	десятичное	перфолента	нет	да	3 оп/сек
ЭНИАК, США, 1945	Электронный	десятичное	ручная коммутация	нет	да	5000 оп/сек
EDSAC, Кембриджский университет, 1949 (первый)	Электронный	Двоичное	«хранимые программы» - прямая запись машинных кодов в память	да	нет	До 15000 оп/сек

После фон Неймана

- В дальнейшем такое двойное использование памяти наоборот стало ограничением, т.к. невозможно из одного модуля памяти одновременно считывать команды и данные.
- На новом витке вернулись к разделению команд и данных. Такая архитектура была названа Гарвардской – по происхождению первой в США ЭВМ МАРК 1 (Гарвардский Университет)