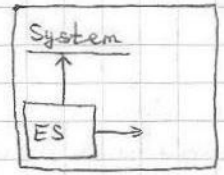
ПОВС

WinDLX
Proteus 6x
ПЛИС + VHDL

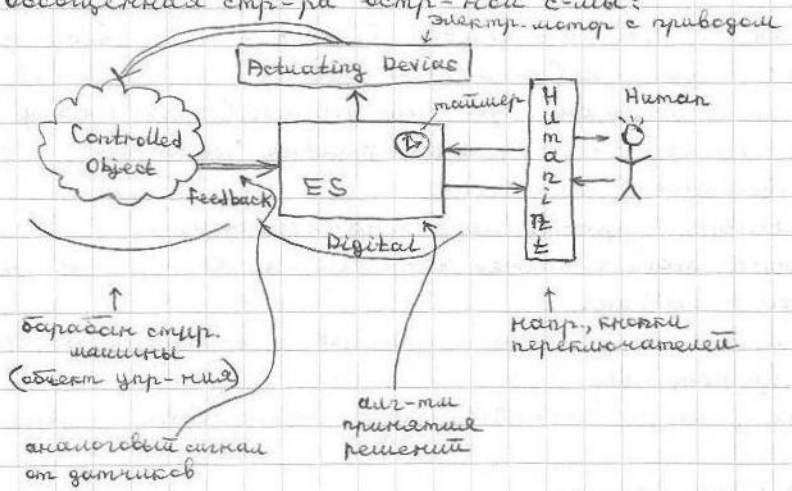
Встроенная система — специал-ная цифровая выч. система, явл-щаяся неотъемлемой частью большей системы, к-ая ей управляет.

Embedded Systems

ES, ВС



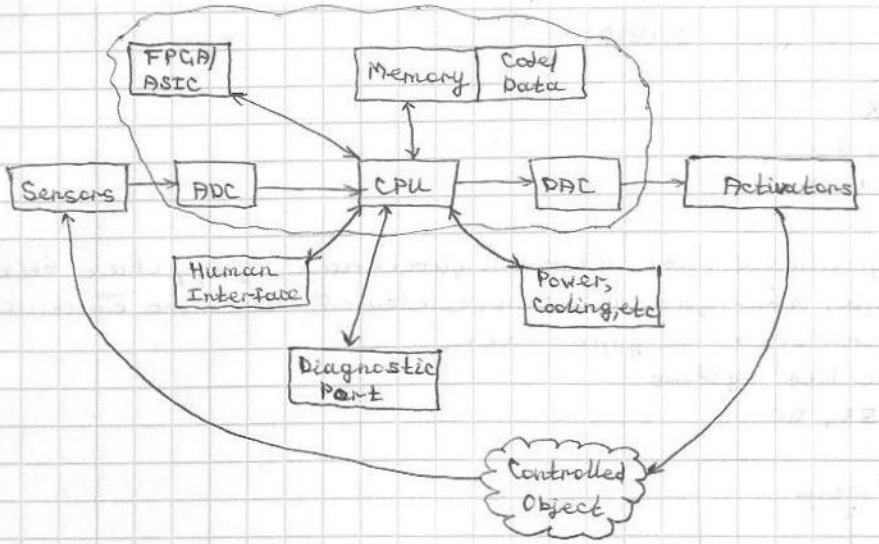
Обобщённая стр-ра встр-ной с-мы:



1) преобр-ть цифровые сигналы в аналоговые
2) осущ-ть различные алг-мы по изменению аналоговых сигналов

Во встроенных системах обр-ка сигналов произв-ся в режимах реального или псевдореального времени.

Уточнённая стр-ра (система) встроенных систем:



CPU осущ-ет реал-цию алг-мов принятия решений. Если алг-мы в пр-ном коде, то необх-ма память для него.

Application Specific Integrated Circuit – если весь алг-м реализован в железе (заказная схема)

Из памяти храним обр-мые данные (есть всегда)

ADC, DAC – преобр-ли Analog to Digital Converter

Field-Programmable Gate Array

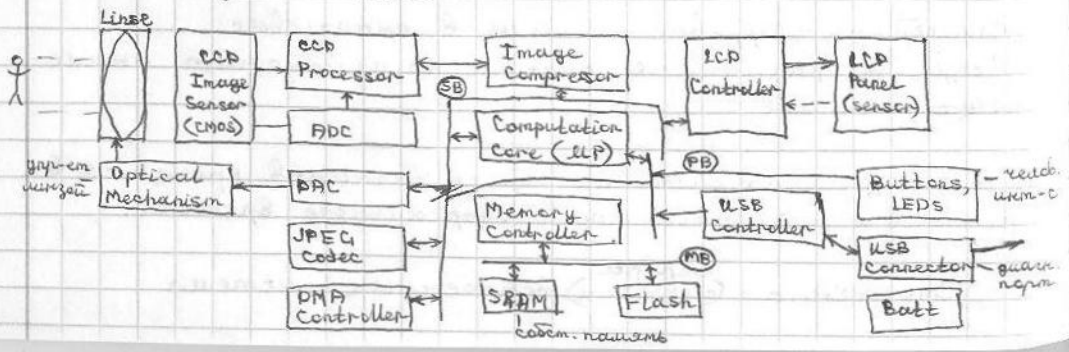
Human Interface – простейшие индикаторы

Абсолютна любая система проект-ся с портом диагностического вывода.

Перепрошивка позволяет изменить алг-мы принятия решений (перепр-вать).

Вместо заказной СБИС лучше поставить прогр-мую.

ES Digital Camera



Всё остальное – размытое цифровое устр-во.

SB – System Bus PB – Peripheral MB – Memory

Контр-р памяти необх-м для считывания инф-ции из памяти или записи инф-ции

Coupled Charge Device (CCD)

Liquid Crystall Display (LCD)

Light Emition Diod (LED)

Direct Memory Access (DMA)

BIST BISD BISR google it

testing diagnostic repairing

Built-in Self reconfig. hardware

Выбор и реализация вычислительного ядра

С т.з. вычислит. ядер м. прим-ся след. подходы:

- выч. центр. ядро на µC
- µP
- soft-P
- embedded-P
- ASIC (заказная интегр. схема)

µC – выч. устр-во на 1 интегр. схеме, включает центр. устр-во, разн. блоки памяти и многие периф-ные блоки (таймеры, ADC, DAC и т.д.). µC имеет усечённый набор инструкций, не имеет встроенной кэш-памяти ни кода, ни данных, RISC, 2х конв-ный. Осн. задача не вычисления, а ф-ции упр-ния периферией. Связан с внешним миром SB и PB.

µP – выч. устр-во на отд. инт. схеме или её части и вкл-ет центр. процесс-ное устр-во с расшир. набором инстр-ций (CISC, VLIW). На нём могут реш-ся любые задачи с один-вой эфф-стью. Встроенная память как правило кэш (инстр-ции данных). Связь с внешним миром с пом. SB; для связи с периферией исп-ся пер-ный контроллер.

System on a Chip (SOC) – система на кристалле; напр., µC, µP (Soft-Processor)

SOC и заказные СБИС – с фиксир-ной арх-рой.

Прогр-ные инт. логич. схемы позв-ют динамически рекон-фиг-ать их аппаратную стр-ру (ПЛИС) — реал-ция любого цифр-го узла любой сложности (огран-ся только аппар-ми ресурсами). Совр. ПЛИС позв-ю реал-ть от сотен тысяч до десятков миллионов настр-мых аппар-ных блоков. Каждый блок м. сод-ть дес-ки триггеров и ...

HDL-языки исп-ся для создания ПЛИС (МС, МР)

IP-component — способ описания МС.

Soft-Processor — реал-ван на ПЛИС, может прогр-ся, имеет память, НО! можно статически изменить его арх-ру. Можно орг-ть многопр-сть, огр-ние — ресурсы ПЛИС.

E-P (Embedded)

ASIC — воплощение полностью в аппаратуре выч-ных или упр-ких алг-мов

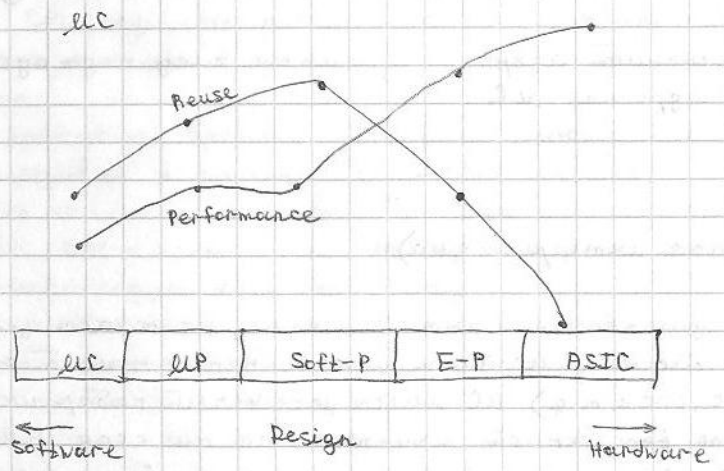


График отобр-ет варианты выбора вычислительного ядра проектировщиками.

Reuse — может ли выч-ное ядро адапт-ся к решению разл. задач (функциональная гибкость).

Soft-P — точка компромисса.

Static reconfig — пример — Atom (Embedded-Системы), стан-ся популярны.

С точки зрения потр-ля встроенные с-мы (с-мы на кристалле) д. удовл-ть след. тр-ниям:

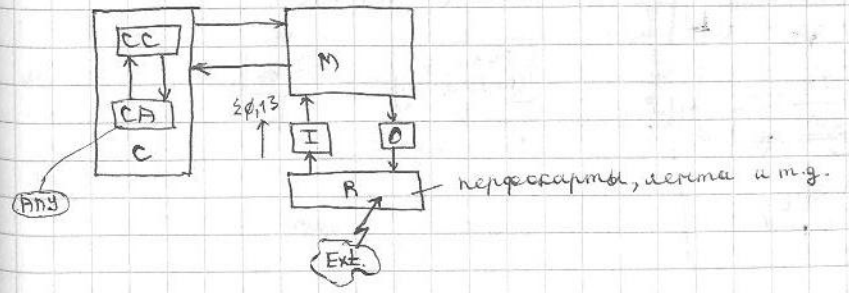
1) высокое быстродействие — эфф-ное вып. задач
2) низкое энергопотр-ние, обусл-ное исп-нием в портатив-ных устр-в, раб-щих на автономных ист-х питания
3) надёжность — польз-ль д.б. убеждён в прав-сти функц-ния системы
4) эксплуатационная гибкость — подраз-ет оперативное изм-ние и дополнение функц-ных возм-стей цифрового устр-ва с участием польз-ля
5) большая степень интеграции — пар-р опр-ет не только число транз-ров на ед. площади кристалла, но также важные хар-ки цифр-го устр-ва и косвенно его физ. габариты
6) приемлемая стоимость — эфф-ость продажи системы на потребительском рынке

Пути повышения быстродействия выч-х ядер:

I.	II.	III.
1) Технология изготовления	1) Исп-ние реконфиг-мой !	3) Многоядерность
2) Повышение сис-темной частоты (понижение температур)		2) Аппаратные ускорители Hardware Accelerators
3) Встраиваемая память PiM — processor in memory		1) Оптимизация исполняемого кода
		0) Правильный выбор архи-тектуры !

Классические вычислительные арх-ры

Стр-ра Фон Неймановского вычислителя



C — control

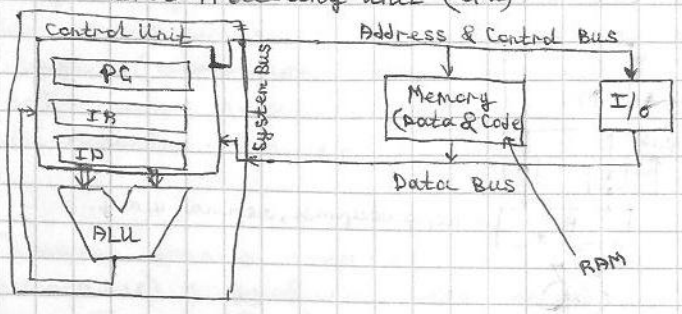
CC – Central Control
CA – Central Arithmetic
M – Memory
I – Input
O – Output
R – Recording Media
Ext – External Device (Human)

1) Все данные и упр-ния, начиная с блоков I и O предст-ны в двоичной СС. Данные и сигналы упр-ния пост-ют через R непоср-но в память, запис-ся инф-ция в нецифровом виде Ext на R. I переводит инф-цию в цифровой вид.
2) Блок C нач-ет послед-ное извлечение упр-щих сигналов из M для согласования работы CC и CA. Затем выбир-ся нач. данные и перед-ся на CA. Рез-т запис-ся обратно в память. Далее выб-ся след. посл-сть упр-ния и т.д.
Рез-т конечного выч-ния также помещ-ся в память, система сигнал-ет об этом и выв-ся через O.

Послед-сть упр-щих (данных) сигналов, к-ая влияет на функц-ние C, наз-ся инструкцией. Посл-сть инструкций – пр-мма (микропр-мма).
Data control в M
Data code
CC → Control (устр-во упр-ния)

Современная интерпретация принципов Фон Неймана при постр. выч. устр-в

Control Processing Unit (CPU)



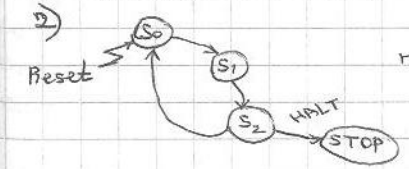
PC – Program Counter
IR – Instruction Register
ID – Instr. decoder
ALU – АЛУ

Счётчик команд указ-ет на инстр-цию в блоке памяти, к-ая д.б. извлечена в блоке упр-ния. Выбр-ная инстр. помещ-ся и хран-ся в IR. ID распознаёт тип выбранной инструкции.

Цикл Фон Неймана

Ф-ние всего выч. ядра следует рассм-ть во времени. Нач. момент система нах-ся в нач. состоянии:
∅ Init, PC = ∅
Исп-мый код расположен в линейном участке памяти
a a+1 a+2 ...
1. устр-во упр-ния выбирает из памяти инструкцию, указ-нную адресом из PC: [PC] ⇒ IR
Fetch – выборка
2. IR ⇒ ID
Decode PC++
На этапе устр-во упр-ния распознаёт тип разпозн-мой инструкции.
3. ID ⇒ ALU
Данные передаются в ALU, инстр-ция выч-ся.
Execute
4. ALU ⇒ Mem (data)
Запись в память
Write
5. Goto 1

Нужна особая инстр-ция окончания работы пр-ммы. На втором этапе распознаем эту к-ду, дальше варианты:
1) 3. Execute PC = ∅
2)



переход в конечное состояние

?

Изм-нию подв-ся не данные, а PC.

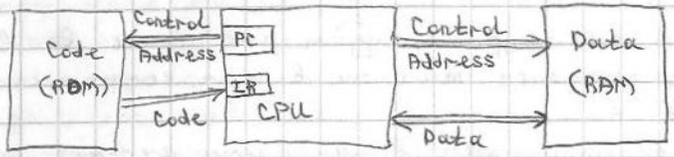
Цикл Фон Неймана — один машинный такт. Этапы имеют свою прод-сть, опр-щую длину такта.

1 и 4 самые длит. этапы.

Разделим память на данных и кода — Гарвардская

Структура Гарвардской архитектуры

Harvard



Можно одн-но извлекать i-ую инструкцию и записывать рез-тат i-1-ой или извлекать операнды i-ой.

ISA — Instruction Set Architecture

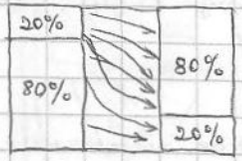
CISC

Complex Instruction Set Architecture

Сложность набора инструкций закл-ся в:

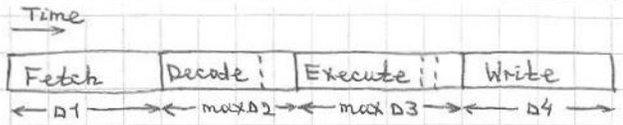
- большом числе инструкций
- разнотипность инструкций
- самое большое число способов адресаций (прямая, косвенная, непоср-ное, автоинкр-ное...)
- различные инстр-ции требуют различной длины машинного такта для исполнения

Закон 80%/20% (Парето) для CISC (усилие — успех)



Дисбаланс CISC-архитектуры

20% инстр-ций исп-ся в 80% кода.



CISC: $\Delta 1 \neq \Delta 2 \neq \Delta 3 \neq \Delta 4$

$$\max_{i=1}(\Delta_i) = \Delta_x$$

Все шаги делаем Δ_x

Каждый шаг цикла Фон Неймана исп-ет работу опред-ных блоков:

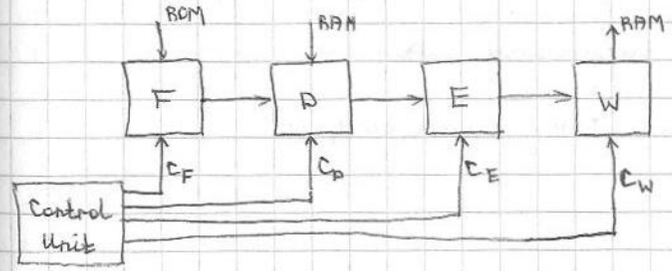
F: Controller ROM, IR, PC

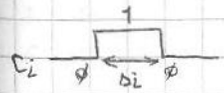
D: ID, IR, RAM Controller

E: ALU

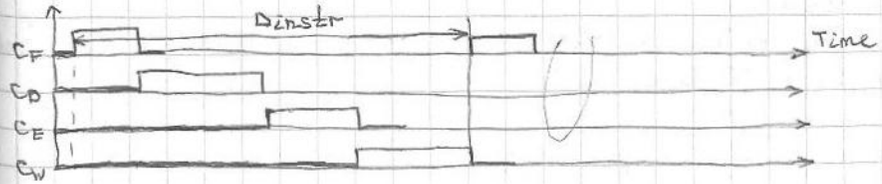
W: RAM Controller

Для корректного исполнения цикла в аппаратуре необх-мо упр-ние данными аппаратными блоками.





Врем. д-ма упр-щих сигналов для орг-ции цикла Фон Неймана:



Импульсы одинаковой ширины. В блоках F, D, E, W введены доп. регистры для хранения промеж. данных

Конвейерная обработка потока инструкций.

Для линейных участков исполняемого кода:

$i=1 \Rightarrow t=4$ $\quad t=4$

$i=2 \Rightarrow t=5$ $\quad t=8$

$i=3 \Rightarrow t=6$ $\quad t=12$

$i=4 \Rightarrow t=7$ $\quad t=16$

$i=M \Rightarrow t=M+3$ $\quad 4M$

конв-р

$$\frac{4M}{M+3} \approx 4$$

Конвейер – специфика устр-ва управления.

Различные варианты ISA

Классические ISA-архитектуры

1) Stack ISA
2) Accumul. ISA
3) GPR ISA (РОН) General Purpose Registers

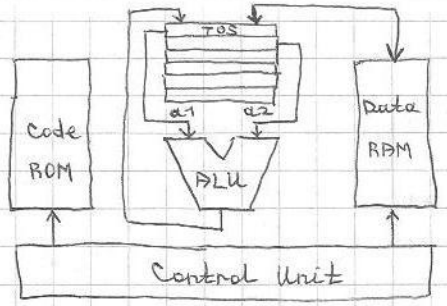
регистры общего назн-ния

Все 3 м.б. и Фон Нейман, и Гарвард, и конв-р.

Stack ISA

За основу возьмём Фон Нейм. арх-ру выч-ля. Память пусть орг-на Гарвардски.

Стр-ра стековой арх-ры:



TOS – top of the stack

$f = ax + b$

Стековая арх-ра подр-ет наличие инструкций push и pop

push adress

push a

RAM[a] ⇒ TOS

push a

push x

pop f

TOS ⇒ RAM[f] $\quad$ пересылка д-х осущ-ся с пом. push и pop

MUL

ADD

MUL

[TOS] * [TOS-1] ⇒ TOS

a1 ↓ a2

$f = ax + b$

1 push a

2 push x

3 mul 4 push b

5 add

6 pop f

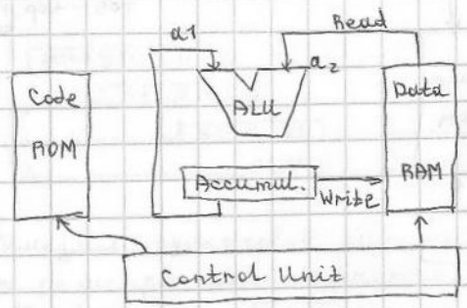
Достоинства арх-ры:

– простота формата и аппаратуры декодирования инстр-ций

Недостатки:

– все данные проходят через 1 регистр – вершину стека (парал-сть невозм-на?)

– трудно понять запись инстр-ций



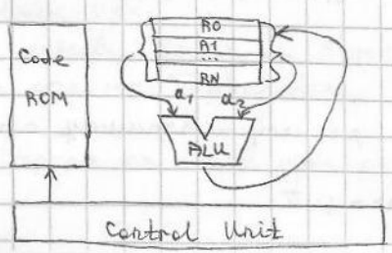
load a — загр. в аккумулятор
RAM[a] => Accum.
store f
[Accum] => RAM[f]
Load-Store ISA — альт-ное название
mul x
[Accum] * RAM[x] => Accum

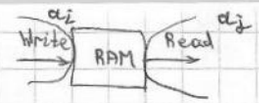
f = ax + b
1 load a
2 mul x
3 add b
4 store f

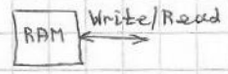
Дост-ва:
- код более читабельный
- мало операндов

Недостатки:
- узкое место — аккумулятор (как и у стековой)
Жива в совр. МС.

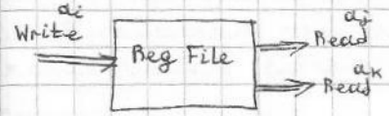
GPR ISA







One-Port нельзя совместить чтение и запись одн-но

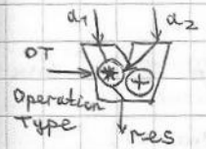


i, j, k даже м.б. равны

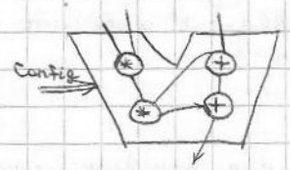
GPR сод-ит регистровый 3-х портовый файл с возм. чтения двух рег-ров одновр-но

1 mul f, a, x
2 add f, f, b

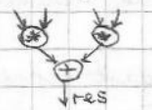
Сложность декодирования, формата инструкций.



4) Reconfigurable ALU

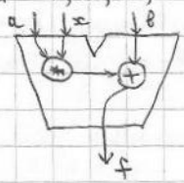


confg.1



confg.2

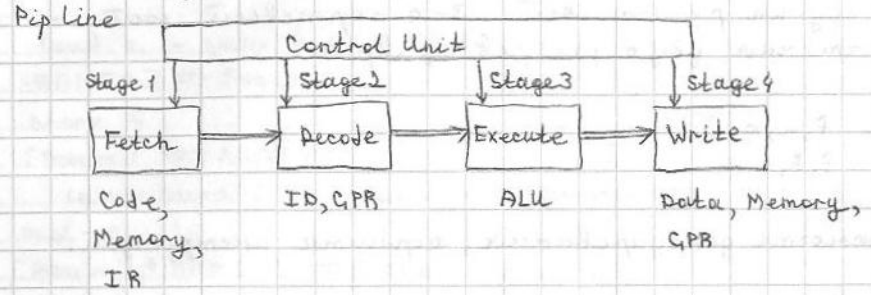
muladd f, a, x, b



Исполнение инстр-ций на реконф-ем ALU:
1) конф-ция блоков внутри ALU
2) исполнение

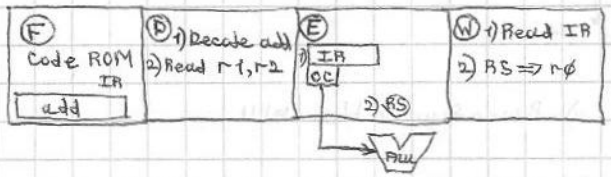
RISC
Reduced Instruction Set Computer

4 ступени кон-ра:



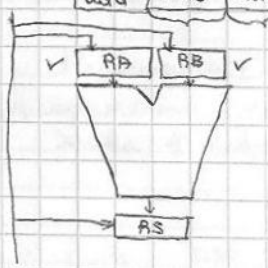
Stage 3 – изм-ние спец. регистров, м.б. даже обр. к памяти

add r0, r1, r2



D По информ-ии коду инстр-ции опред-ся формат инстр-ции.

add	r0	r1	r2
	res	adr1	adr2

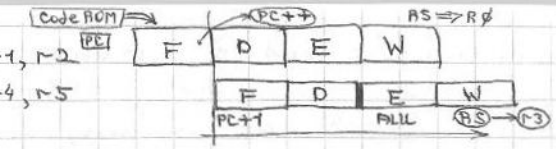


– прогр-но недоступные теневые регистры ALU

Фиксация нужна для корректного вып-ния операций (проблема времени).

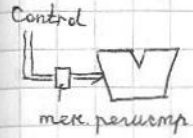
E 1) Из IR берем код инстр-ции OC и исп-ем его для указания ALU, какую операцию необх-мо исполнить.
2) Рез-тат вып-ния фиксир-ся в теневом регистре RS

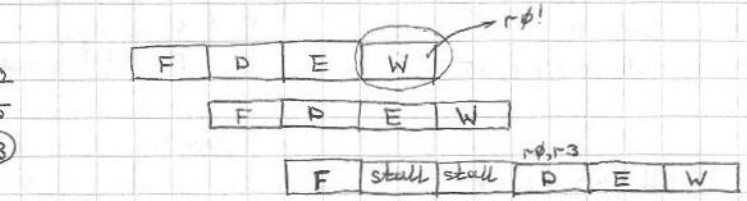
W 1) читаем регистр инстр-ций для нах-ния реального физ адреса рез-та



Как только инстр-ция выбрана PC++.

Предположим, что зн-ние PC инкрем-ся автом-ки в конце стадии выборки (для дальнейшего).





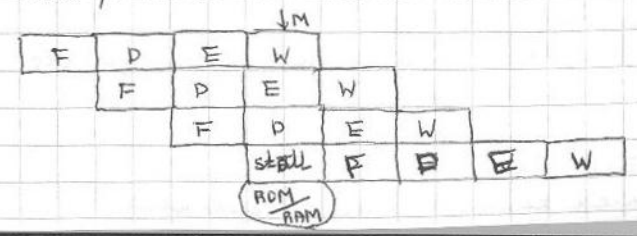
Конфликты по данным тут были.
Data Conflicts
Hazards

Типы конфликтов по данным:
1) Read After Write (в примере) – RAW
2) WAW } обращение к внешней памяти требует более 1 ступени
3) WAR } кон-ра reg ← Mem
reg → Mem

Load r0, M(r1) – составить конфликт

RAR не конфликт, т.к. данные неизменны.

Помимо к-тов по данным выделяют по упр-нию и стр-ные.
Control Structural
Стр-ный приводит к принуд-ному простою к-ера из-за стр-ных аппар-ных особ-стей вычисл-ного ядра.
Пример:



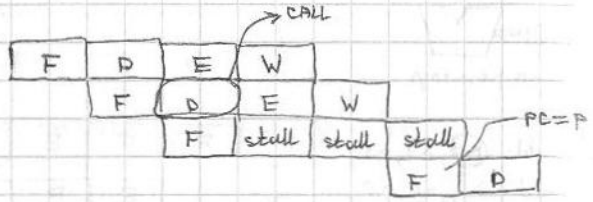
Смысл устранения к-тов в минимизации времени выч-ния процессов.

Решение стр-ных к-тов, как правило, в изм-нии стр-ры вычислителя.

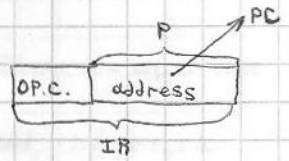
Стр-ный к-т м. закл-ся в невозм-сти пар-ной обр-ки ALU запросов (длинные операции).

К-ты по упр-нию

Пример:



pc: add r0, r1, r2
pc+1: CALL P
pc+2: sub r3, r4, r5
⋮
P: mul r2, r3, r4



Кроме линейных участков кода м.б. нелинейные, к-ые изменяют PC (инстр-ции перехода).

- безусловный переход JUMP PC ← address
- условные JE PC←address PC=PC+1
- вызов CALL
- прерывание IRQ
- возврат из проц-ры RET

PC: CALL:
1) PUSH PC+1
2) PC <= new
3) Execute

Конв-р не исполнит CALL пока не законч-ся все инстр-ции до CALL.

Fetch (PC^{new}) — при этом конв-р заново начинает функц-ть с новым PC зн-нием

RET:
1) Исп-ся как обычная инстр-ция
2) Берём из стека PC^{old}
3) Сбрасываем конв-ер, продолжаем с PC^{old}

Методики минимизации конвейерных конфликтов

В реальных пр-ссах простои неизбежны; методики — оптимизация кода по быстродействию — прогр. и аппар.

Прогр.: осн. на модификации графа выч. процесса исполняемого кода.

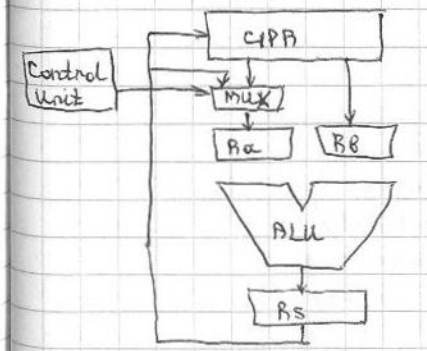
1) пересмотр карты ПОЖ
2) предск-ние переходов
3) разв-ние циклов
4) inline-функции

Пр-ные методики:
1) исп-ние эквивал-ных инстр-ций

Аппаратные методы осн-ны на внесении стр-ных преобр-ний в аппаратную сост-щую соотв-щих ступеней.

К апп-ным отн-ся:
1) быстрая пересылка данных (Fast Forwarding)
2) исп-ние мультипортовых рег-ров данных (Multiport REGFILE)
3) прим-ние суперскал-ных подходов (мк-во ALU)
4) реконфиг-ние (Reconfig. Comput.)

Fast Forwarding закл-ся в модификации ALU, к-ая подразумевает быструю пересылку результ-щих данных из Rs не только в GPR, но и на один из входов ALU



MUX — мультиплексор

add R0, R1, R2
add R3, R4, R5
sub R6, R0, R3

		E R5	W	R5→R0	
F	R1, R2	add	R01		
	F	R4, R5	add	R3	
		F	Ra	Ra, R0	sub

Значение из SS запис-ся, как в R0, так и в Ra.
Быстрая пересылка сраб-ет только в мом. пот-ного останова и закл-ся в пересылке рез-та на 1 из входов ALU. Этим экономится время, необх-мое для записи рез-та W. Наличие быстрой пересылке не отраж-ся на проц-ре записи в CPR.
Наличие БПД не означает полной ликв-ции простоя и пр-ет исп-ния с прогр-ми методами оптимизации.

Пр-ный метод (Code Transformation)

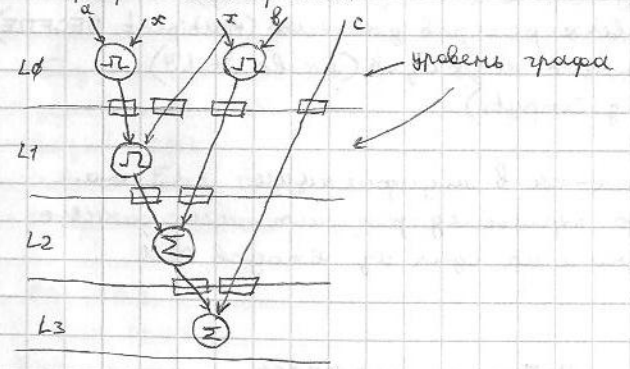
add R2, R1, R0
add R3, R2, R1 ⇒
add R4, R3, R1

F	R1, R0	add	R2						
	F	stall	stall	R2, R1	add	R3			
		F	stall	stall	stall	stall	R3, R1	add	R4
1	2	3	4	5	6	7	8	9	10

6 → 10 12 – обычный фон Неймана

$f = ax^2 + bx + c$

граф. выч. пр-сса:



R2 = R1 + R0
⇒ R3 = R2 + R1
R4 = R3 + R1

⇒

R2 = R1 + R0
R3 = 2R1 + R0
R4 = 3R1 + R0

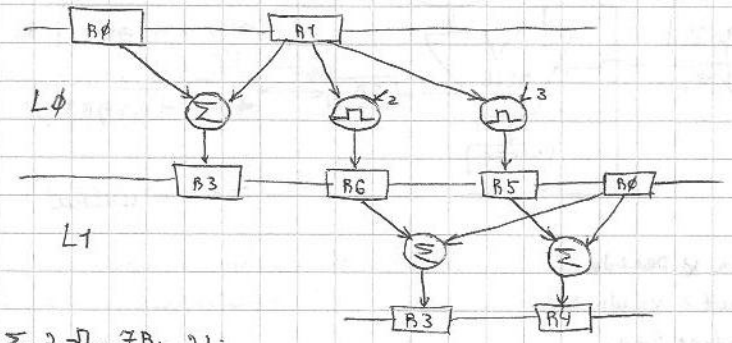
⇒

R5 = 3R1
R6 = 2R1
R2 = R1 + R0
R4 = 3R1 + R0
R3 = 2R1 + R0

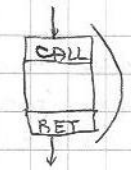
1) MUL R5, R1, 3
2) MUL R6, R1, 2
3) ADD R2, R1, R0
4) ADD R4, R5, R0
5) ADD R3, R6, R0

F	R1	MUL	R5				
	F	R1	MUL	R4			
		F	R1, R0	ADD	R2		
			F	R5, R0	ADD	R4	
				F	R6, R0	ADD	R3
1	2	3	4	5	6	7	8

Граф выч. пр-сса без ограничения на число опер. блоков и регистров

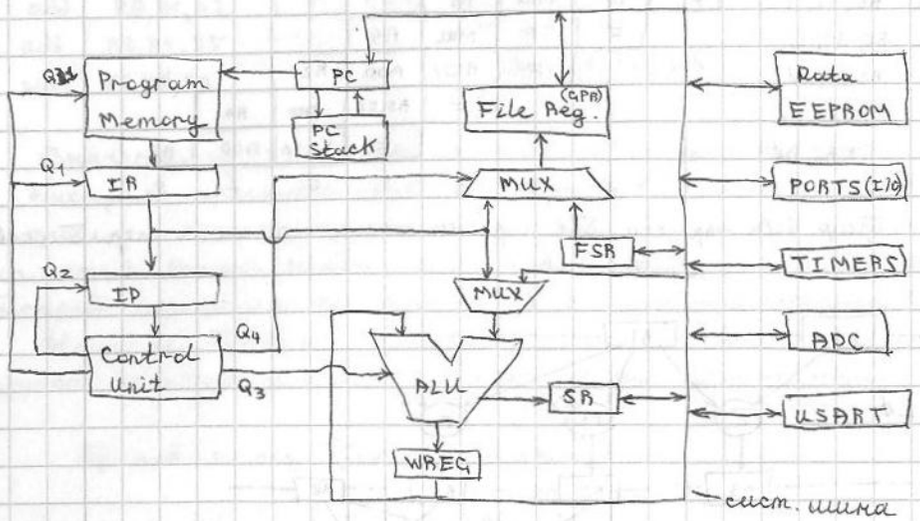


1Σ, 2Π, 7R_L, 2L_i



Micro Chip
PIC16F 84X
RISC, 2pipeline
EEPROM
35 инструкций

Внутренняя структура микроконтроллера



(Q_1, Q_2) = Fetch & Decode
$(Q_3\ Q_4)$ = Execute & Write
FSR – File Select Reg.
SR – Status Reg.
WReg – Write (Result) Reg.

Аккум. принцип: WREG – акк-р (он же источник 1-го операнда ALU). Второй операнд м.б. зн-ем из рег-ра инстр. (const) или инф-ция от сист. шины д-х.

Wreg можно модифицировать пр-но.

SR уст-ет свои биты в зав. от событий (напр., прерывание).

FSR исп-ся для адресации.

File Reg. – GPR

Все устр-ва, подкл. к сист. шине, явл-ся прогр-но доступными. Устр-ва упр-ния выр-ют 4 упр-щих синхросигнала Q_i.

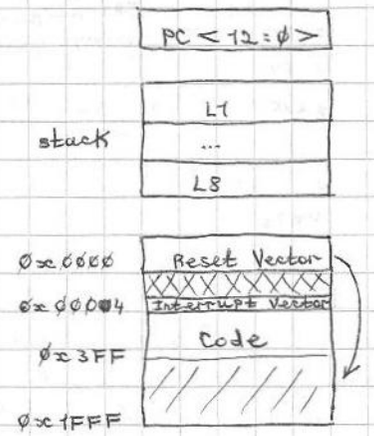
Q_4 опр-ет, куда д. запис-ся рез-тат

PC	PC+1	PC+2

$F.D_{PC}$	E, W_{PC}	
$E.W_{PC-1}$	F, D_{PC+1}	

Возможен только к-т по упр-нию (call).

Орг-ция памяти программ



– 13-и разр-ный

физ-ки 1 канал обр-ки прерываний

Макс. объём памяти – 4 килослова (4K по 13).
Обычно 1 килослово (1K) макс. исп-ся 0x3FF

```
0x0000   Goto Start
   ⋮
         ORG 0x10
         Start:
0x0010   MOVLW 0x3F      – инф-ция аккум-ра const
         Literal  аккум.
```

Память данных вкл-ет:
- 68 однобайтных ОЗУ
- спец. регистры (SR)
- упр-щие рег. периф. устр-в

Они нах-ся в едином адр. пр-ве памяти данных. Всё адр. пр-во разделено на 2 банка.

Логич. стр-ра памяти данных

0x00	INDF	INDF	0x80
1	TMR0	OPTION	0x81
Bank 0 2	PCL	PCL	0x82
3	STATUS	STATUS	0x83
4	FSR	FSR	0x84
5	PORTA	TRISA	0x85
⋮			
0x0c	GPR (68 штук)	GPR	0x8c

Bank1

INDF – спец. рег., исп. для косв. адр.
TMR0 – рег. таймера
TRISA – рег. спец. упр. портом

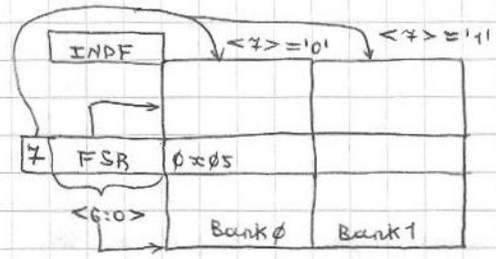
Режимы адресации

При исп. тек. инстр. важным явл-ся адресуемый банк данных. Зн-ние адр-мого банка опр-ся 5 битом STATUS<5> или

RP0 = "0" ⇒ Bank0
"1" ⇒ Bank1

```
BSF 3,5                 bit set file
BSF STATUS,RP0          в 3 бит 5-го файла уст-ть 1 ; Bank1

BCF 3,5 ⇒ BCF STATUS,RP0  ; Bank0

BCF STATUS,RP0
MOVLW 0x11
MOVWF 0x05              // 0x85 то же самое
 addr. file
WReg = 0x11
WReg ⇒ [0x05]

BSF STATUS,RP0
MOVLW 0x22
MOVWF 0x05
Bank1
WReg = 0x22
WReg ⇒ 0x05
```

Indirect Access – косвенная адресация



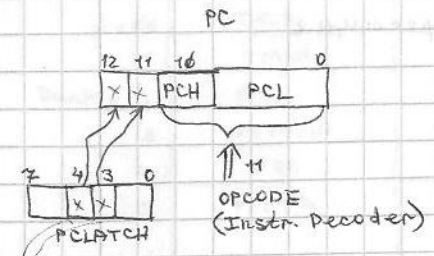

```
MOVLW addr
MOVWF FSR
MOVF INDF,0        – в акк-р зак-ся зн-ние некого файла; но
 WReg = addr         если это файл INDF, то зак-ся зн-ние,
 FSR = addr          адресуемое FSR
 WReg ⇐ [FSR]
```

Пример косвенного чтения

```
MOVLW addr
MOVWF FSR
MOVLW data
MOVWF INDF
 FSR = addr
 WReg ⇐ data
 INDF ⇐ WReg
 [FSR] ⇐ WReg
Bank_i
MOVLW 0x6          WReg = 6
MOVWF FSR          FSR=6
MOVLW 0xFF         Wreg = 255
MOVF INDF          [6] ⇐ 255      PORTB/TRISB
                   косв. адр. нач-ся автом-ки, после записи
Bank_i             в INDF сеанс закан-ся
```

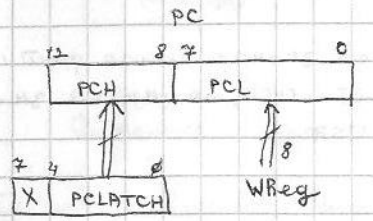
Упр-ние счётчиком команд

CALL, GOTO, RETURN
PCL, PCLATCH – 2 спец. рег.



Есть возм-сть безусл. перехода в пределах 1К.

01 – во 2-ую страницу jmp



Пример пр-ного упр-ния PC

```
0x10   MOVLW pointer   → WReg = pointer = 2
11     CALL Table      → PC:stack ← 12, PC <= 20
12 ·   WREG = TABLE[pointer]
 ⋮
20     ADDWF PCL,F     → PCL = 23
21     RETLW 'B'
22     RETLW 'S'
23     RETLW 'U'
24     RETLW 'I'
25     RETLW 'R'
```

Table-Read

сложение WReg с PCL и рез-т в PCL

возврат со зн-нием

– храним данные в памяти кода

Перед исп-ем инстр. по адресу 20 акк-р сод-жит значение pointer (в д.сл. 2). Зн-ние PCL уже равно 20. В мом. когда уже исп-ся инстр. зн-ние счётчика PCL будет уже автоинкр-но и равно 21 ⟹ после исп-ния PCL = 21 + point = 23 – иск-ное создание ситуации безусл. перехода.

Прогр-ние встроенной EEPROM-памяти

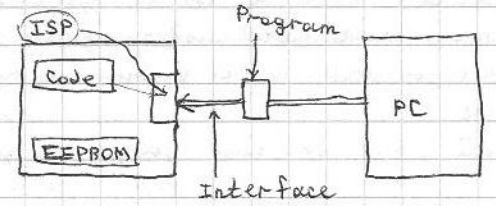
Энергонез-мая

64 байта

Исп-ние:

1) ROM (ПЗУ)
2) Writelog

Прогр-ть память можно на стадии изгот-ния всего МК



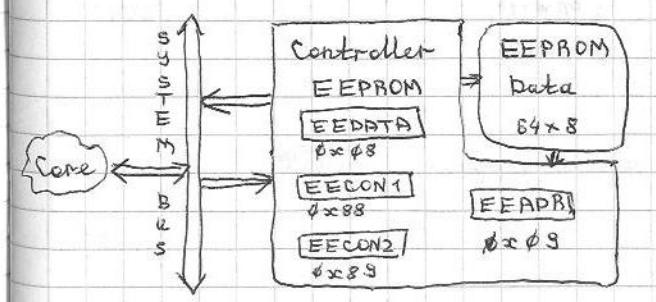
3 способа:

1) окно в MPLab
2) исх. код пр-ммы

```
org   0x2100
de    "BSUIR", 0
org   опр-ем смещение в памяти данных
```

0 (0x2100) – 'B'
1 (0x2101) – 'S'

3) непоср-но при функц-нии МК



EEDATA – регистр данных при осущ-нии операций записи и чтения

EECON1 и 2 – конфиг-ные рег-ры МС, задающие режимы функц-ния памяти, опр-ют зн-ния упр-щих

бит при сов-ние операций и хранят битовые флаги рез-тов операций
EEADR – адрес

EECON1 (Øx88) Bank1

7	6	5	4	3	2	1	0
X	X	X	EEIF	WRERR	WREN	WR	RD

0 – RD – явл-ся упр-щим битом, к-ый инициирует операцию чтения, если он 1 – операция чтения инициирована
1 – WR – если WR = 'Ø' после того, как началась проц-ра записи, это флаг окончания проц-ры записи; если запишем туда 1, то автом-ки нач-ся запись
2 – WREN – упр-щий бит, к-ый разр-ет или запр-ет запись when = '1' allows write
3 – WRERR – бит флага успешности проц-ры записи; '0' – проц-ра записи закончена и прошла успешно
4 – EEIF – бит флага прерывания Interrupt Flag; окончание проц-ры записи – внутреннее прерывание, флаг уст-ся автом-ки с наст. прерывания

Рассм-м примеры процедур чтения и записи байта:

EEPROM Read

```
BCF STATUS, RPØ          ; Bank0
MOVLW Address            ; WReg ⇐ Address
MOVLW EEADR              ; EEADR ⇐ W
BSF STATUS, RPØ          ; Bank1
BSF EECON1, RD           ; PD = '1'
BCF STATUS, RPØ          ; Bank0
MOVF EEDATA, W           ;
```

EEPROM Write

```
BCF STATUS, RPØ
MOVLW Address
MOVWF EEADR
MOVLW Data
MOVWF EEDATA
BSF STATUS, RPØ
BCF INTCON, GIE          ; Interrupt Control  Global Interrupt Enable
BSF EECON1, WREN         ; Write Enable
MOVLW   Øx55
MOVWF   EECON2
MOVLW   ØxAA
MOVWF   EECON2
BSF     EECON1, WR       ; Write!
```

– нельзя прерывать
контрольная послед-сть; подг-ем контроллер для перепрогр-ния EEPROM

```
BSF INTCON, GIE
```

Если не пишем обработчик окончания проц-ры записи, то после инич-ции записи необх-мо ожидание окончания записи (напр. EEIF = '1', потом WRERR = '0'). Если пишем, то разрешаем все прерывания и потом по адресу 4 обработчик.

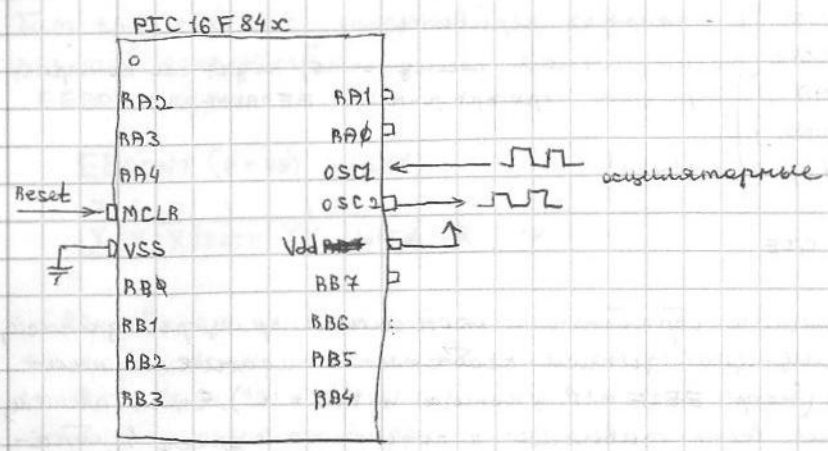
```
BankØ   MACRO
  BCF STATUS, RPØ
ENDM

BankØ

ADD3  MACRO result, arg1, arg2
  MOVLW arg1
  ADDLW arg2
  MOVWF result
ENDM

V EQU Øx2Ø
ADD3 V,2,3
```

Порты ввода/вывода

RØ … RA4 } I/O

RA4 – специфич. (Ø по умолчанию; вх. сигнал синхр. внутр. таймера TØCKI

Все разряды м.б. сконф-ны независимо друг от друга.



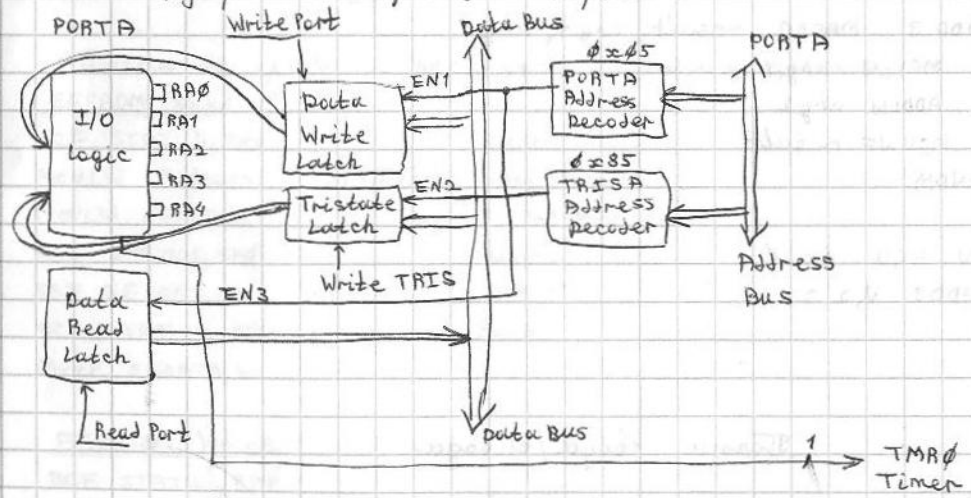
RA_i 0–4

RB_j 0–7

Могут выр-ть и специф. роль.

Могут упр-ть периф. устр-вами (Output); принимать инф-цию (Input); специф. (синхр-ция, напр.)

Внутренняя орг-ция порта A



Write Port
Write TRIS } from Control Unit
Read Port

Операция записи в PORTA изм-ет появление на адресной шине адреса, а на шине д-х – данные

A.B. ⟹ 0x05 D.B. ⟹ data (WReg)

Сраб-ет дешифратор и посылает сигнал EN1 или EN3 на соотв-щие рег-ры

Control Bus ⟹ Write Port = '1' – EN1

EN1 позв-ет захватить данные с D.B. и сохр-ть их в регистре data write.

Захват данных актив-ет одну из 3 защёлок: запись – Data Write Latch; output-порт – I/O logic на вывод; чтение – и I/O logic на вход, то все данные запис-ся в Data Read Latch.

Tristate Latch хранит упр-щие биты для каждого разряда порта (RA_i – Output/Input).

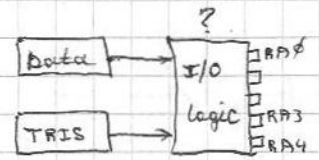
Для упр-ния портом предв-но надо указать конфигурацию соотв-щих разрядов порта и только после этого I/O.

	7	6	5	4	3	2	1	0
PORTA	X	X	X	d	d	d	d	d
				↑	↑	↑	↑	↑
TRISA	X	X	X	0/1	0/1	0/1	0/1	0/1

0 – Output

1 – Input

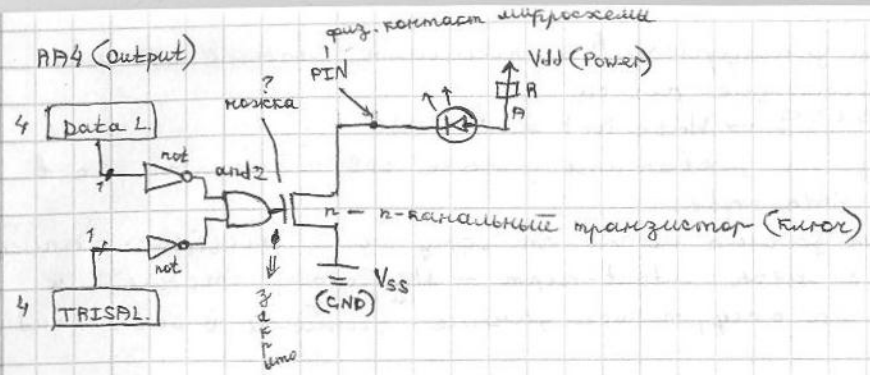
Внутр. стр-ра логики разрядов порта A



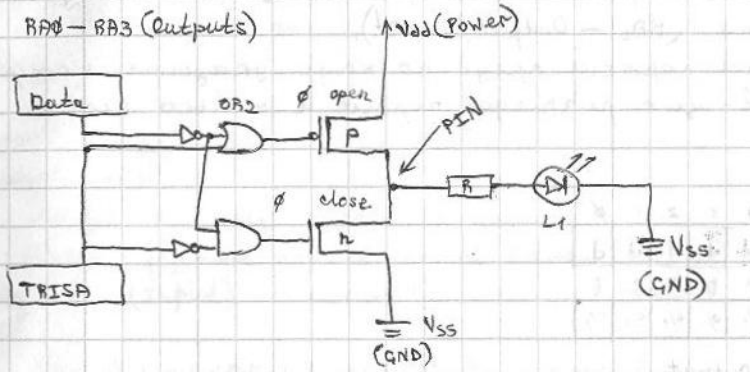
RA0 – RA3 имеют одинаковую схемотехнику, RA4 – упрощённую ⟹ разное упр-ние.

Подключение светодиодов к разрядам порта A (LED – Light Emitting Diode)

RA4 (Output)

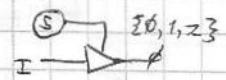


RA0 — RA3 (Outputs)




```
CLRF   PORTA
BSF    STATUS, RP0     ; Bank1
MOVLW  0
MOVWF  TRISA           ; PORTA = 0   TRISA = 0   outputs
BCF    STATUS, RP0     ; Bank 0
MOVLW  0x1F
MOVWF  PORTA           ; PORTA = '11111'
```

RA4 0 на n-кан. транз. => экв-но $\frac{1}{T}$ (разрыв) — не горит! и L1 горит! (в RA0 — RA3).



S=0 O <= I
S=1 O <= 'z'

```
CLRF   PORTA
BSF    STATUS, RP0
CLRF   TRISA
BCF    STATUS, RP0
CLRF   PORTA
```

RA4 : 1 — горит!

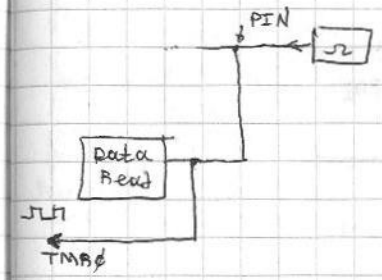
RA0 — 3

OR2 1 close

AND2 1 open — не горит!

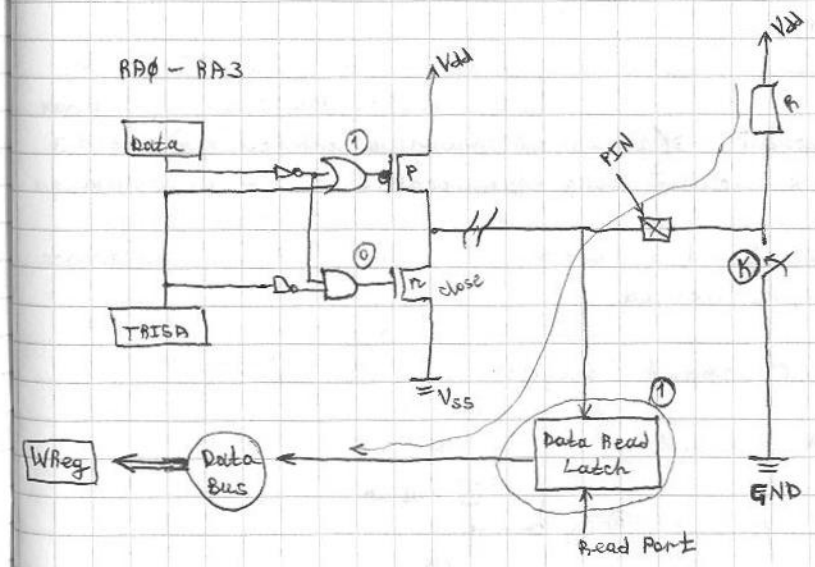
При конфиг-ции RAi как выходных для i = 0 — 3 исп-ся положит. логика упр-ния, а для i = 4 — отриц-ная.

Для Input RA4:



1 в TRISA блокирует предыд. схему (всегда 0).

RA0 — RA3



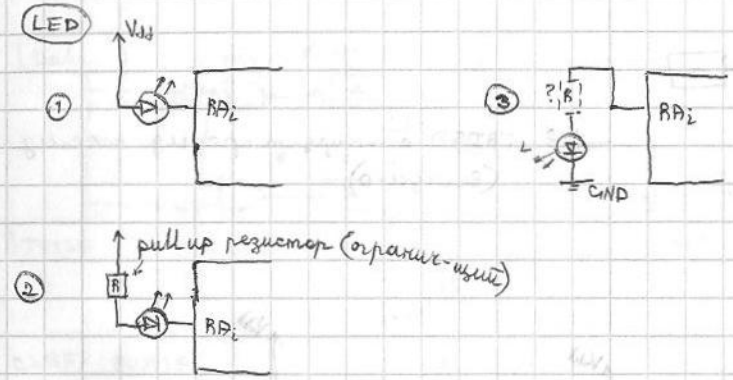

```
BSF    STATUS, RP0
MOVLW  0x1F
MOVWF  TRISA ; '11111'
BCF    STATUS, RP0
MOVF   PORTA, 0
```

W = ?

Если ключ разомкнут, то потенциал даст 1.
после TRISA = '11111' — 1.
После исп-ния MOVF PORTA, ф значения попадут на шину и после сигнала Read Port — в аккум-тор.

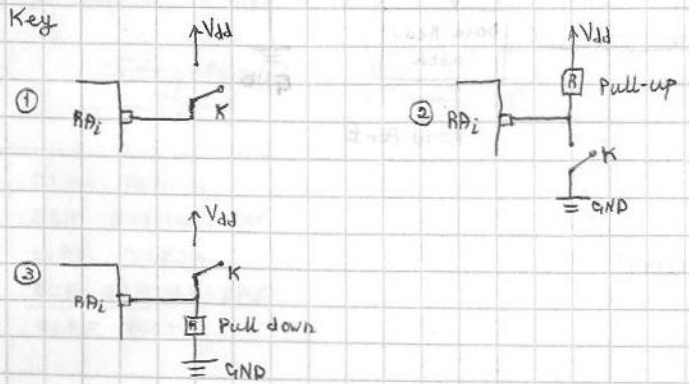
Если замкнут — 0 (низкое напряжение).
Т.о., отриц-ное упр-ние или негативная логика (закрыт-о, открыт — 1).

Сущ-ют разл. схемы подкл-ния прост. устр-в I/O инф-ции.



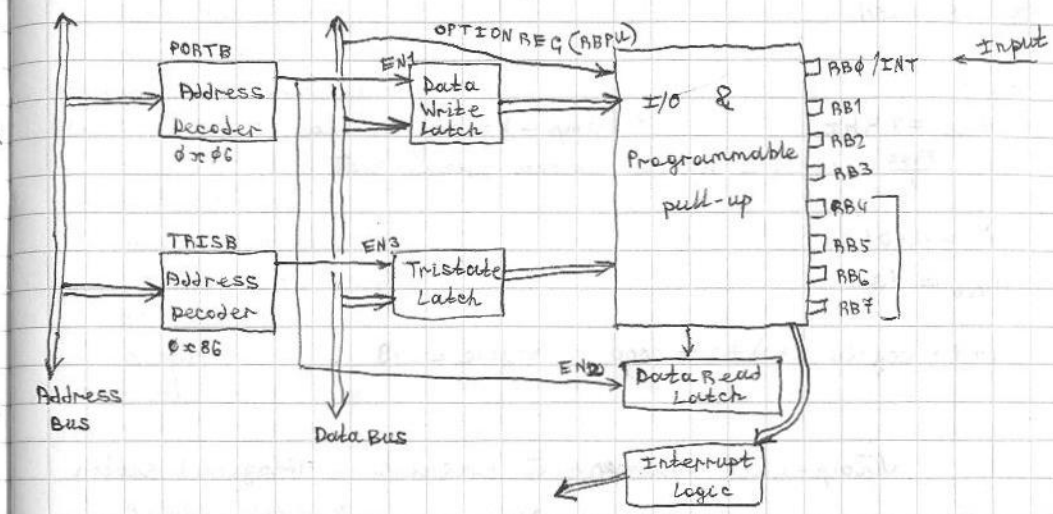
Самый безопасный — 3 (горит от энергии контроллера), но в 3 слабо горит, т.к. малый ток; из плюсов 3 ещё — положит. логика упр-ния.
Самый небез-ный — 1.
Нед-ток — отриц. логика.

Взар-ты подкл. устр-в ввода:

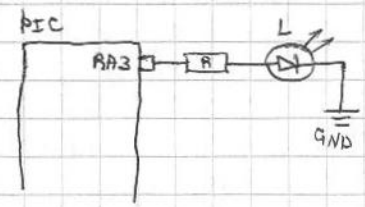


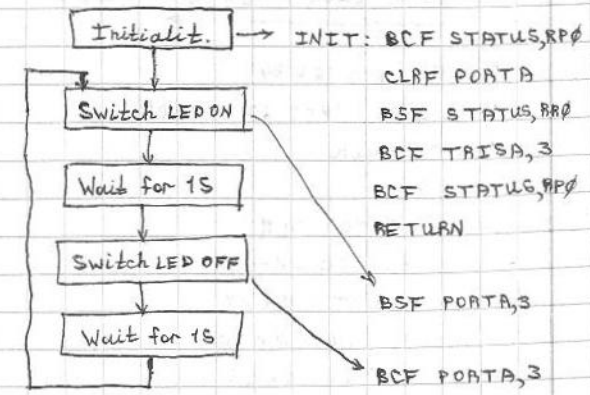
1 — самый небезоп-ный
3 бережёт контроллер, т.к. чаще всего ключи разомкнуты.

Стр-ра порта B
(все как RA0 — RA3)



Каждый разряд м. исп-ть подтягивающие регистры при конф-нии на Input (OPTION RBPU).
I и O как и в A. Кроме: RB0 кроме I/O явл-ся источником прерывания от внешнего устр-ва.
RB4 — RB7 также явл-ся ист-ми прерываний при изм-нии хотя бы одного знач-ния на этих разрядах.





```
WAIT 1S:
(1)   MOVLW vcycle
(1)   MOVWF vpause
L0
(2)   DECFSZ vpause
(2)   GOTO L0
(2)   RETURN
```

OSC1

$F_{sys} = 4\,KHz$ 1 инстр. — 4 сист. такта

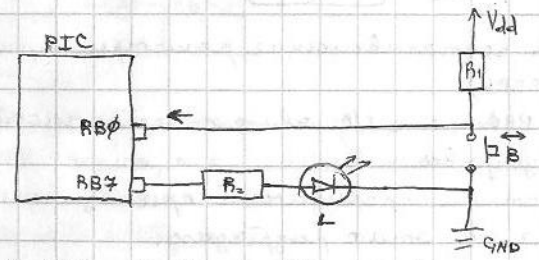
$\frac{F_{sys}}{4} = 1\,kHz$ — обр-ка потока инстр-ций

$\Delta_i = 0{,}001\,s$

$\Delta_{1000} = 1\,s$

$1 + 1 + vcycle(2+2) + 2 = 1000$ $vcycle = 249$

Прогр-ние триггерной кнопки Triggered Button



B = OFF B = ON B = OFF B = ON

L = OFF L = ON L = ON L = OFF

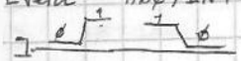
```
LIST p=16F84
#include "PIC16F84.INC"
GOTO Main
ORG 4
HALT: GOTO HALT
constant LED = 0x80
constant BUT = 0x01
BUT_OLD EQU 0x20
BUT_CUR EQU 0x21
LED_STA EQU 0x22
XRES    EQU 0x23
```

```
Main:
    CLRF PORTB
    BSF STATUS,RP0
    MOVLW 0x01
    MOVWF TRISB
    BCF STATUS,RP0
    CLRF BUT_CUR
    CLRF LED_STA
Reset: CLRF BUT_OLD
Loop: CALL BUT_STATUS
      ;W=0  BCS OFF
      ;W=1  BCS ON
    MOVWF BUT_CUR
    BTFSS BUT_CUR,0
    GOTO Reset
    CLRW
    MOVF BUT_OLD,0
    XORWF BUT_CUR,0
    MOVWF XRES
    BTFSS XRES,0
    GOTO Loop
BUT_STATUS:
    BTFSC PORTB,0
    RETLW 0
    RETLW 1
    CALL CLED
    MOVF BUT_CUR,0
    MOVWF BUT_OLD
    GOTO Loop
CLED:
    MOVLW LED
    XORWF LED_STA,1
    MOVF LED_STA,0
    MOVWF PORTB
    RETURN
```

Обработка прерываний

4 типа события, к-ые м. прервать поток инстр-ций:

1) Write into EEPROM
2) TMR0 overflow
3) Изм-ние кода на входных портах RB4–RB7
4) Event RB0/INT



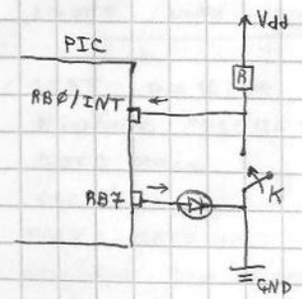
Упр-ние механизмом обр. прер. осущ-ся маник-нием регистром INTCON $\begin{cases} 0x0B \\ 0x8B \end{cases}$

7	6	5	4	3	2	1	0
GIE	EEIE	T0IE	INTE	RBIE	T0IF	INTF	RBIF

Enable — Flag (EEIF)

GIE – Global Interrupt Enable (1 – разр-ся все прер-ния, указ-ные в посл. 4 разрядах; 0 – запр-ся).
EEIE – разр-ет прер. 1-го типа (EEPROM)
T0IE – разр. прер. от внутр. таймера
INTE – разр. RB0
RBIE – RB4–RB7

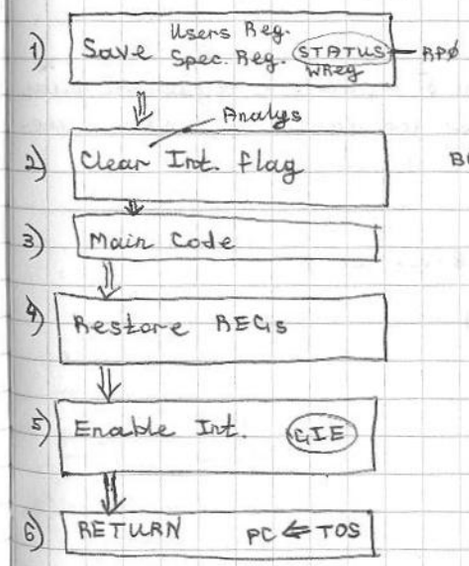
Флаги прер. устан-ся автом-ки по наст-нии соотв-щего события (их сброс вручную осущ-ся).
Вектор прерывания один (по адресу 4).




```
ORG 0x0
GOTO Main
ORG 0x4
GOTO LED_STATUS
constant led_stat = 0x80
ORG 0x10
Main:
  CLRF PORTB
  BSF STATUS, RP0
  MOVLW 0x01
  MOVWF TRISB
  BCF STATUS, RP0
  CLRF INTCON
  BSF INTCON, INTE
  BSF INTCON, GIE
x:
  ⋮
  goto x
LED_STATUS:  BCF INTCON, GIE
  MOVLW led_stat
  XORWF PORTB, 1
  BCF INTCON, INTF
  BSF INTCON, GIE
  RETURN
```

PC [PC+1] ⇒ TOS

Основные положения по сост-нию обраб-ков прерываний



BCF INTCON, IF

При возврате из обр-ка желательно:

```
BSF INTCON, GIE
RETURN
  ||
RETFIE
```

Событие на линии RB0 м.б. 2 типов

RB0 0 1 0

источники прерываний на RB0

0x81 OPTION_REG, INTEDG

BCF 0x81, 6 ; RB0

BSF OPTION_REG, INTEDG ; default = 1

Управление регистрами при обр-ке прерываний

```
Save:    MOVF   VAR1, 0
         MOVWF  VAR1_tmp

Restore: MOVF   VAR1_tmp, 0
         MOVWF  VAR1
```

Упр-ние акк-ром

```
Save:    MOVWF  W_tmp
Restore: MOVF   W_tmp, 0
```

При восст-нии зн-ния акк-ра м.б. изменён нулевой флаг регистра STATUS, к-ый автом-ки устан-ся в 1 при нулевом значении в акк-ре.

```
WReg:
Restore*:
        SWAP   W_tmp, 1   ; 1   рез-т в сам рег-р
        SWAP   W_tmp, 0   ; 2   в W
```

W_tmp = 0x5F

WReg: 1) W_tmp = 0xF5

2) WReg = 0x5F

STATUS:

```
Save:    SWAPF  STATUS, 0
         MOVWF  STATUS_tmp
Restore: SWAPF  STATUS_tmp, 0
         MOVWF  STATUS
```

Пример:

```
Save:    BTFSS  STATUS, RP0
         GOTO   S_BANK0
         BCF    STATUS, RP0
         MOVWF  W_tmp
         SWAPF  STATUS, W
         MOVWF  STATUS_tmp
         BSF    STATUS_tmp, 1
         GOTO   INT_CODE

S_BANK0: MOVWF  W_tmp
         SWAPF  STATUS, 0
         MOVWF  STATUS_tmp
INT_CODE:
         ⋮
Restore:
         SWAPF  STATUS_tmp, 0
         MOVWF  STATUS
         BTFSS  STATUS, RP0
         GOTO   Restore_W
         BCF    STATUS, RP0
         SWAPF  W_tmp, 1
         SWAPF  W_tmp, 0
         BSF    STATUS, RP0
         RETFIE
Restore_W:
         SWAPF  W_tmp, 1
         SWAPF  W_tmp, 0
         RETFIE
```

Обр-ка нескольких прерываний

1) Decoding (1)

→ CALL → [1]

Prior. → [2]

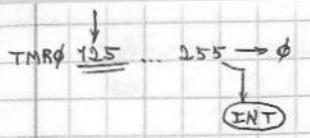
→ [3]

2) Decoding (1 & 2)

Внутр. стр-ра и прерывания таймера TMR0

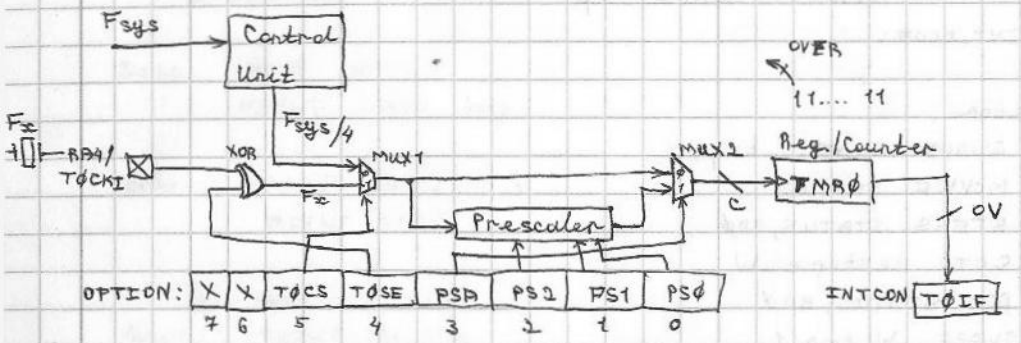
0 ... 255 → 0

INT



Исп-ся для прогр-ной реал-ции временных интервалов. Также для написания приложений псевдо реального времени (диспетчеризация).

Стр-ра таймера TMRØ



TMRØ – обычный 8-и разр. двоичный счётчик (начально все 0, сигнал C инкр-ет на 1).

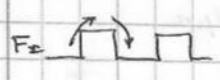
Синхр-ция таймера.

1) $\frac{F_{sys}}{4}$

2) внешний источник F_x (к порту RA4)

3) Prescaler – предварит. делитель частоты (м. делить входную частоту на к-ент).

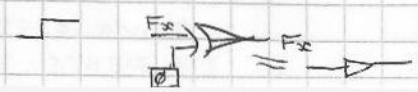
Мультиплексоры служат для выбора 1 из 2 ист-ков и включить Prescaler.

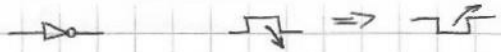


XOR опр-ет ↗ или ↘ (фронт или спад)

пр-ным способом можно удвоить F_x – чередовать ↗ и ↘

TØSE = Ø



TØSE = 1 

TØSE – бит упр-ния фронтами сигнала для ист-ка RA4.

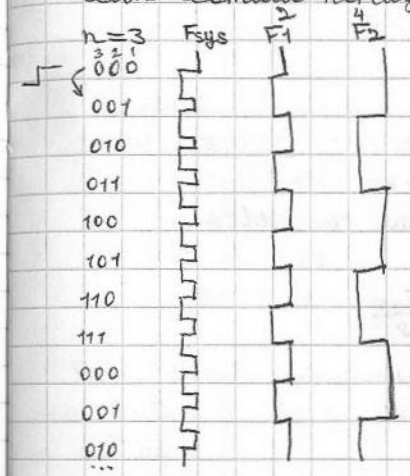
TØCS = 'Ø' $F_{sys}/4$ = '1' F_x

MUX1 выб-ет ист-к

MUX2 упр-ет Prescaler

PSA = 'Ø' Prescaler не работает ⟹ $F_{sys}/4$

Если хотим понизить ~~к-ту~~ частоту, то PSA = '1'

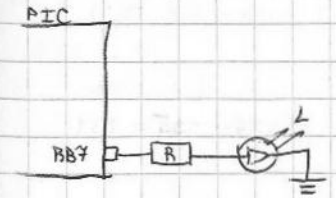


К-ты деления зад-ся 3 разрядами PS2–PSØ

PS2	PS1	PSØ	R_K F_{sys}
Ø	Ø	Ø	2
	⋮		4
			...
1	1	1	256

$\frac{F_{sys}}{1024}$ – предел

Сам счётчик тоже делитель частоты. Т.о., включив Prescaler и исп-вав сам счётчик частоту F_{sys} можно знач-но уменьшить для обр-ки прер. от таймера.



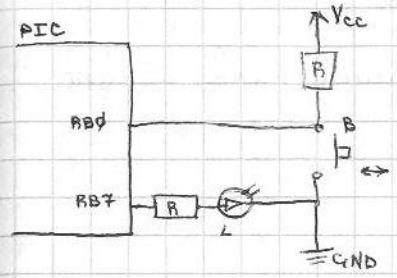
```
ORG 0x0
GOTO Main
ORG 0x4
GOTO LED_STATUS
Constant c_led_sta = 0x80
ORG 0x10
Main:   CLRF PORTB
        CLRF TMR0
        BSF STATUS, RP0
        MOVLW 0x0
        MOVWF TRISB
;настройка таймера
    BCF OPTION_REG, 5   ; выбор Fsys/4
    BSF OPTION_REG, 3   ; будем исп-ть Prescaller
    BCF OPTION_REG, 2
    BCF OPTION_REG, 1   } R=2      Fsys/8
    BCF OPTION_REG, 0
    BCF STATUS, RP0
    CLRF INTCON
    MOVLW XXX
    MOVWF TMR0
    BSF INTCON, T0IE
    BSF INTCON, GIE     ; МС готов обр-ть 1 событие: перен. TMR0
```

обработчик:

```
LED_STATUS:
    MOVLW c_led_sta
    XORWF PORTB,1
    MOVLW XXX ?
    MOVWF TMR0
    BCF INTCON, T0IE
    BSF INTCON, GIE
    RETURN
End
```

$F_{sys} = 1\,kHz$ $F_L = 1\,Hz$ XXX – ? $= 256 - 125 = 131$

Обработчик двух событий.



Основной обр-чик:

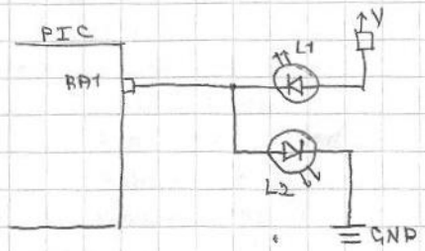
```
INTH:   BTFSC INTCON, INTF
        CALL BUTTON
        BTFSC INTCON, T0IF
        CALL LED
        RETFIE
```

Если 2 соб. произошли одн-но (нажата кнопка и перен-лся таймер), то первым обр-ся событие по кнопке, потом от таймера.

```
BUTTON:
    MOVLW 0x1
    XOR v_but_sta,1
    BCF INTCON, INTF ; сброс своего флага
    RETURN

LED:
    BTFSS v_but_sta, 0
    GOTO EXIT_L
    MOVLW c_led_sta
    XORWF PORTB,1
    MOVLW 0xAA ; произв. конст.
    MOVWF TMR0
    BCF INTCON, T0IF; сброс флага
EXIT_L:
    RETURN
```

Исп-ние мин. числа интерфейсных линий для упр-ния светодиодами

Пример упр-ния 2 светодиодами 1 инт. линией.



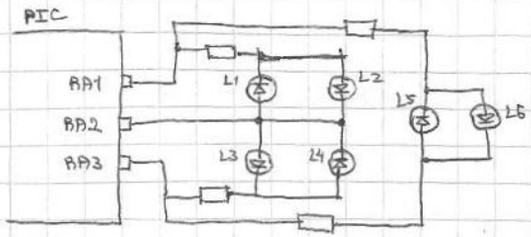
①
```
BSF  STATUS, RP0
BCF  TRISA,1
BCF  STATUS, RP0
BCF  PORTA,1          ; L1 горит, L2 не горит
```

②
```
BSF  STATUS, RP0
BCF  TRISA, 1
BCF  STATUS, RP0
BSF  PORTA,1          ; L2 горит, L1 не горит
```

③
```
BSF  STATUS, RP0
BSF  TRISA,1
BCF  STATUS, RP0      ; оба горят, т.к. факт-ки разрыв цепи
```

④ RA1 как выход, на к-ый подавать 0-1-0-1... с такой частотой, чтобы они выглядели негорящими (учитываем латентность светодиодов; $t_{лат.} \approx 5$ ms).

Данная методика позв-ет подключать больше 2 диодов. Пример упр-ния 6 диодами с пом. 3 инт. линий.



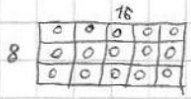
RA_i (out) $\in \{0,1\}$						
RA_i (in) $\in \{Z\}$						
RA1	0	1	Z	1	0	0
RA2	1	Z	Z	1	0	1
RA3	Z	0	Z	1	0	0
L1	+	–	–	–	–	+
L2	–	–	–	–	–	–
L3	–	–	–	–	–	+
L4	–	–	–	–	–	–
L5	–	–	–	–	–	–
L6	–	+	–	–	–	–

Подобное упр-ние не очень хорошо, т.к. ярко гореть не будут. Лучше упр-ть не анодами и катодами, а транзисторными ключами электронными.

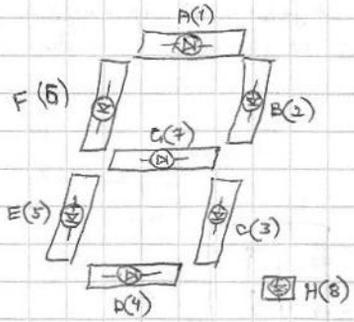
Помимо прост. светодиодов исп-ся: - семисегментные индикаторы (7 Segments LED) 8 светодиодов + точка. Они м.б. бинарные (надо 8 линий для упр-ния) и двоично-дес. (надо 4).

- линейки светодиодов

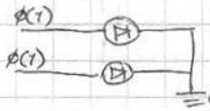
- светодиодные матрицы (панели)



7-сегментные индикаторы

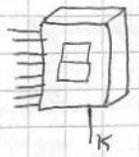


Для упр-ния необх-мо и дост-но 8 упр-щих линий. Они все могут иметь общий катод:



или $\phi = const$, но GND экономит провод

М.б. с общим анодом (соед. или разрыв с землёй транзист. ключами).



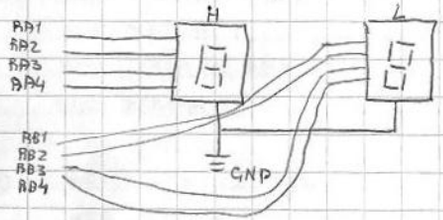
— двоично-дес. — имеет декодер из $(I_4, I_3, I_2, I_1) \Rightarrow$ $(A, B, \dots, C)$

A B C D E F G
1 1 1 1 1 1 0

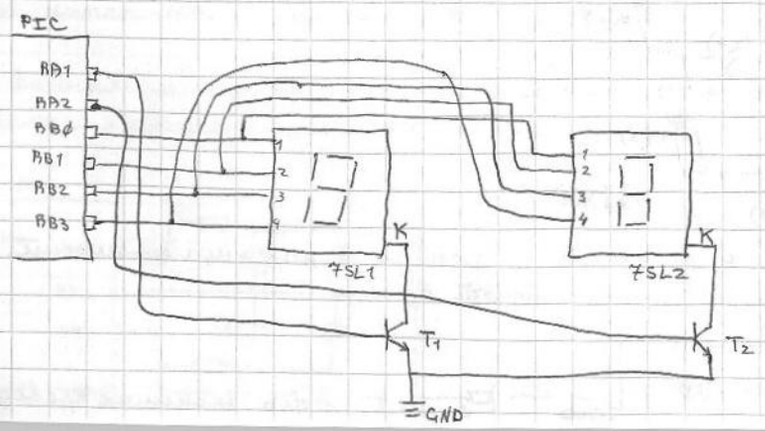
0 0 0 0 — (0)

1111 — F(15)

Семисегм. индикаторы



Мультипл-ние подр-ет исп-ние общих информаций для передачи символа, а адресация осущ-ся упр-нием их общими анодами или катодами.



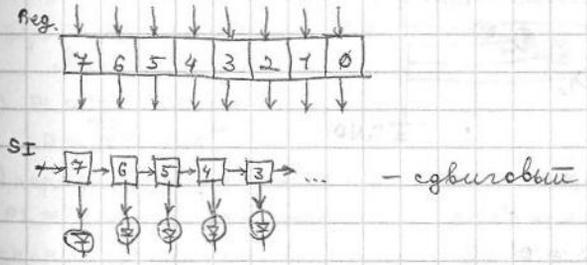
Алг-тм упр-ния:

1) Запис-ем код символа в порт B S1 → PORTB
2) Активируем 7SL1 путём T1 on
3) Wait for latency, чтобы разгорелся
4) Прекращение свечения T1 off
5) Запис-ем код S2 → PORTB
6) Активируем T2 on
7) Wait for latency
8) T2 off
9) Goto 1

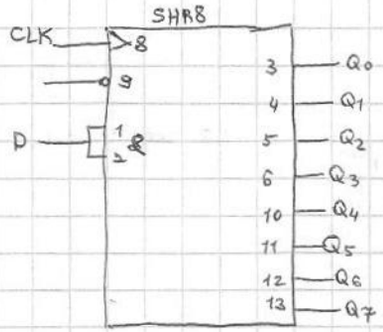
Период повторения данной проц-ры д.б. таким, чтобы комплекс-ть эффект мигания индикаторов. Можно упр-ть яркостью свечения обоих индикаторов.

Исп-ние регистров сдвига для упр-ния индикаторами

Широко исп-ся регистры сдвига для преобр-ния последних кодов в парал-ные.



Пример сдвигового регистра
Philips Semclon. 74HC164



CLK — синхрон.

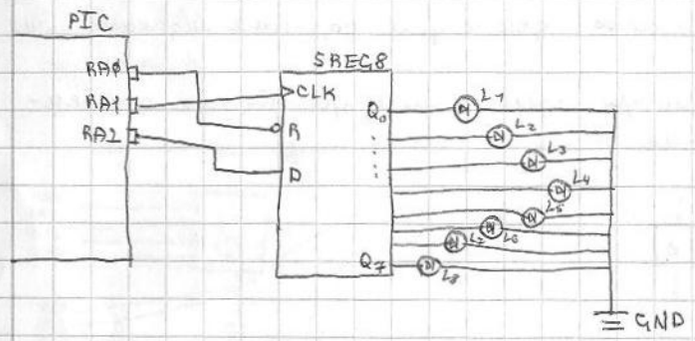
R — reset (общ-ние)
D — сигнал вх. данных
$\langle Q_0 - Q_7 \rangle$ — парал. выход регистра

Функц-ние сдвиг. рег.: в нач. мом. предп.:

1) CLK = '0' R = '1' D = '0' Q = "XXXXXXXX"
2) R = '0' Q = "00000000" R = '1'
3) CLK = '1' Q = "00000000"
4) D = '1' CLK = '0'
5) CLK = '1' – осущ-ся захват зн-ния входа D с его записью в старший значимый разряд сдвиг. рег-ра (Q_7)
   Q = '10000000' CLK = '0'
6) CLK = '1' Q = '11000000' CLK = '0'

Для полного зап-ния $Q_0 \dots Q_7$ необх-мо 8 сигналов синхр. CLK.

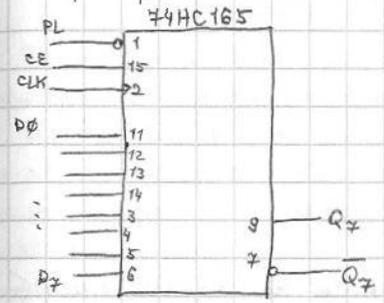
Пример подкл. у упр-ния в светодиодами




```
Main:   BSF PORTA,0     ; R='1'
        BCF PORTA,1     ; CLK='0'
        BCF PORTA,2     ; D='0'
        BSF STATUS,RP0  }
        CLRF TRISA      } outputs
        BCF STATUS,RP0  }
Loop:
        BCF PORTA,0     ; R='0'
        BSF PORTA,0     ; R='1'
        BSF PORTA,2     ; D='1'
        BSF PORTA,1     ; CLK='1'          L8 загор-ся
        BCF PORTA,1     ; CLK='0'
        CALL Wait
        BCF PORTA,2     ; D='0'
        MOVLW 0x07
        MOVWF VLP       ; VLP = 7
L0:     BSF PORTA,1     ; CLK='1'
        BCF PORTA,1     ; CLK='0'          бегущий огонь
        CALL Wait
        DECSFZ VLP
        GOTO L0
        GOTO Loop
```

Также обр. ф-ция — преобр. парал-ного кода в послед-ный.

Пример:

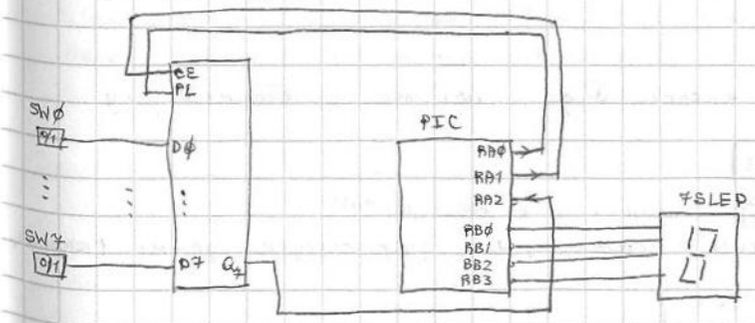


Сдвиг м. осущ-ся пост-вом фронта CLK либо при пом. упр-щего сигнала CE (Clock Enable). PL – сигнал парал. загр. (Parall. Load)

Алг-тм упр-ния:

1) PL = '1' CE = '1'
   D = 100111 00 (MSB … LSB)
2) перед загр. зн-ния D в рег-стр Q_7 и $\overline{Q}_7$ нах-ся в неизв. х сост.
   PL = '0' – регистр открыт. для фикс. нужного зн-ния PL = '1' – зн-ние D зап-ся в рег-р. Q_7 = MSB(D7) = '1' (у нас)
3) CE = '0' ⇒ CE = '1' – сдвиг (задний фронт CE = 0?)
4) после сдвига Q_7 = [MSB − 1] = 0. Необх-мо 7 периодов синхр. CE для захвата 8-ми D (MSB доступен без сдвига).

Пример исп-ния:



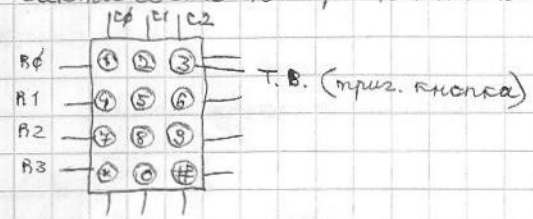
Необх-мо сост-ть упр-щую пр-мму для МС, к-ая будет отображать двоичный код пер-лей SWi на 7SLED.

Этапы:

1) Иниц-ция сдвиг-го рег-ра путём захвата зн. SWi
2) Посл-ный сдвиг разрядов через порт RA2 в кач. перем.
3) Отрр-ем пер-ную на 7SLED.

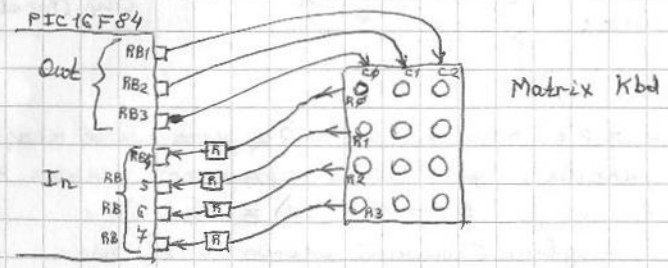
Матричные клавиатуры

Фактически набор контактов



При нажатии замыкаются линии на пересечении (ост. разомкнуты).

Схема подключения:



Прерывание по изм-нию кода на RB4 – RB7

RB1	RB2	RB3

Для код-ния кнопки будем исп-ть семибитный код.

C0	C1	C2	R0	R1	R2	R3
0	0	1	0	0	0	0
0	1	0	0	0	0	0
1	0	0	1	0	0	0

– скан-код клавиши "1"

Условие декод-ния – нажатия и удержание одн-го 1 кнопки.

one-hot $\begin{bmatrix} 00\ldots01 \\ 00\ldots10 \\ \ldots \\ 10\ldots00 \end{bmatrix}$

Нажмём кнопку "1" (C0; R0).

0 10 0001 – "0"

Алг-м прогр-ния матр. клав.:

1) имеем проц-ру генерир-ния цикличн. one-hot послед-сти
2) после подачи текущего зн-ния one-hot послед-сти читаем RB4–RB7 (или же исп-ем обр-к прерывания по изм-нию RB4–RB7 – лучший вар-т)
3) анализируя зн-ния RB4–RB7 сост-ем скан-код нажатой кнопки (вес по Хеммингу д.б. равен 1).
иначе ошибка произошла – напр., 2 кнопки сразу.

Скан-коды м.б. ужаты путём перекодирования (напр., номера разрядов (00, 01, 10 и 00, 01, 10, 11).

Необх-мо учит-ть:

1) ограничения по нажатию
2) мин. время нажатия кнопки (расч-ть частоту генния one-hot).

Лучше исп-ть прер-ния по RB4–RB7 для избежания колизий одн-го или послед-го нажатия.

Необх-мо учит-ть ситуацию ненажатия.

Устр-ва вывода инф-ции LCD

Буквенно цифровые LCD класс-ся:

1) кол-во отобр-мых символов и строк (обычно от 8 до 16 и более символов в 1 или 2 строках)

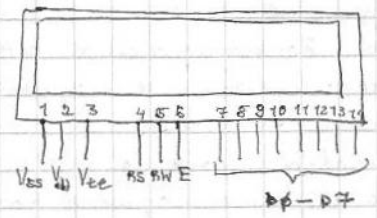
Сложные устр-ва, для их упр-ния исп-ют контроллеры, вх-щие в LCD-модуль.

LCD 2 lines

LM016M (Hitachi HD44780)

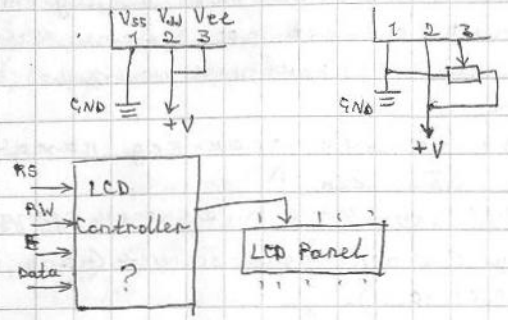
2 строки по 16 символов. 1 символ – матрица 8x5.

Можно заменять зн-ты матрицы.

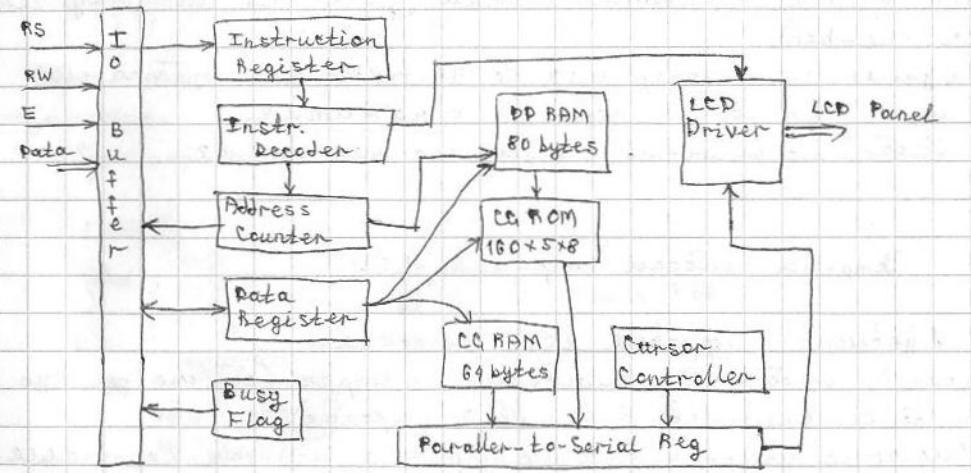


RB1 – RS RB2 – RW RB3 – E RB4 – RB7 – D11 – D14

V_{SS} – GND V_{dd} – Power V_{EE} – back light RS – Register Select



Стр-ра LCD Controller



DDRAM – Display Data RAM
CGROM – Character Generator ROM

RS = 0 ⇒ данные – код отобр-мого символа
RS = 1 ⇒ данные – код инструкции

Тек. позиция отобр-мого символа хран-ся в Address Counter. Его зн-ние автом-ки инкрем-ся при получении кода символа для отобр-ния.

Можно изменить Address Counter прогр-но.

CGRAM для польз-ских символов. – энергозавис. память.

Символ курсора – отд. объект для упр-ния и отобр-ния (в любом месте, большой, мигающий...).

Каждая инстр-ция тр-ет опр-ного времени исполнения (зад-го произв-лем) ⇒ можно не исп-ть Busy Flag.

Контроллера режимы:

1) режим Init инициализации (опр-ние типа инт-са, вкл./выкл. подсветки, перемещ. курсора в нач. поз., опр-ние CGRAM)
2) режим Data передачи данных (для DDRAM или CGRAM).
3) Instr. режим передачи инструкций

Δ_{min} – 40 μs Δ_{max} – 4 ms
мили микро

При исп-нии полубайтного режима при инич-ции необх-мо указать особую инстр., после чего все данные и инстр. будут перед-ся по 4 бита со старшего полубайта.

Delay_1ms
Delay_2ms
Delay_4ms
Delay_40us

```
#define bank0
        bcf   STATUS,RP0
#define bank1
        bsf   STATUS,RP0
RS equ 1
RW equ 2
EN equ 3
LCD_buf equ 0x11
LCD_tmp equ 0x12
; LCD instructions
constant LCD_CD = 0x01 ; clear Display 2ms
constant LCD_CH = 0x02 ; Cursor Hom 2ms
constant LCD_ON = 0x0C ; LCD On 40us
constant LCD_OFF = 0x08 ; Display Off 40us
constant LCD_CN = 0x0E ; Cursor On 40us
constant LCD_CB = 0x09 ; Cursor Blink 40us
constant LCD_2L = 0x28 ; LCD has 2 Lines 40us
constant LCD_4B = 0x20 ; 4-bit interface 40 us
constant LCD_L1 = 0x80 ; select 1 line  –"–
constant LCD_L2 = 0xC0 ; select 2 line  –"–
```

RB7	6	5	4	3	2	1	RB0
D7	D6	D5	D4	E	=0	RS	

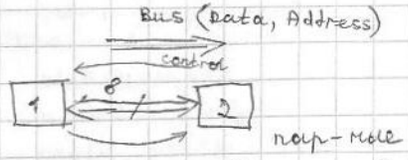
```
;W - операнд (парт).
LCD_cmd
  clrf LCD_buf
  movwf LCD_tmp
  andLW b'1111 0000'
  iorwf LCD_buf, 0          ; inclusive OR (просто OR)
  call LCD_wr
  swapf LCD_tmp, 0
  andLW b'1111 0000'
  iorwf LCD_buf, 0
  call LCD_wr
return
;LCD_wr - передаёт данные из W, формируя импульс E  1ms g.b.
 LCD_wr:
  clrf PORTB
  movwf PORTB
  call Delay_1ms ; необх-мо для норм. сигналов на входе уп.
  bsf PORTB, EN
  call Delay_1ms
  bcf PORTB, EN
 return
; для g-x то же самое + RS = 1
LCD_dat
  bsf LCD_buf, RS
   ...

LCD_init:
  bank1
  clrf TRISB
  bank0
  clrf PORTB
  call Delay_4ms; ] LCD PowerUp!
  movlw LCD_4B
  call LCD_cmd      } cmd
  call Delay_40us
  movlw LCD_ON
  call LCD_cmd
  call Delay_40us
  movlw LCD_2L
  call LCD_cmd
  call Delay_40us
```

```
  org 0x10
Main:
  call LCD_init
  movLw 0x41
  call LCD_dat
  call Delay_40us
macro
   movLw 0x61
   call LCD_dat
   call Delay_40us

LCDD macro Ldata
   movLw Ldata
   call LCD_dat
   call Delay_40us
endm

LCDC macro Lcmd
  movLw Lcmd
  call LCD_cmd
  call Delay_40us
endm

   LCDC 0x40     ; CGRAM
   LCDD b'10001'
   LCDD 0x00
   LCDD 0x00
   LCDD 0x00
   LCDD 0x09
   LCDD 0x09
   LCDD 0x09
   LCDD 0x1F
   LCDC 0x80
   LCDD 0x40
```

Инт-сы встроенных систем

Инт-ы:

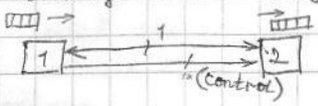
- пар-ные
- послед-ные



пар-ные

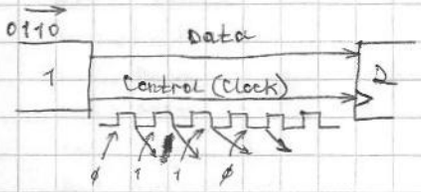
- односторонние (однонапр.) — полудуплексный
- двунаправленные (полнодуплексный)

послед-ные: полудупл. только

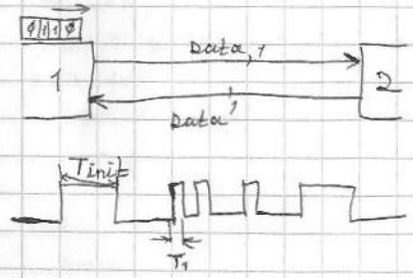


послед. передача пакетов

- синхронные
- асинхронные



— синхр.



удерж-ся в 0 — пробел
1 — символ

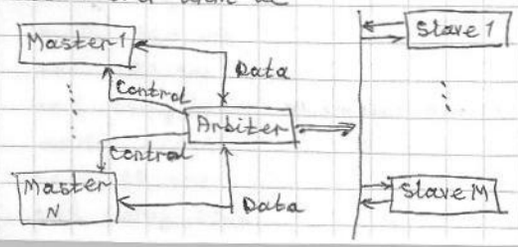
асинхр. послед. инт-сы синхр-ют свою работу по соотв-щим временным интервалам.

По типу общения 2 цифровых устр-в выд-ют:

- Master - устр-во ведущее (упр-ет)
- Slave — ведомое (упр-ся)

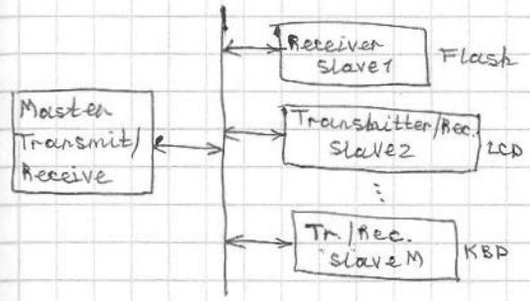
Виды подключений:

- Multi Master инт-ес



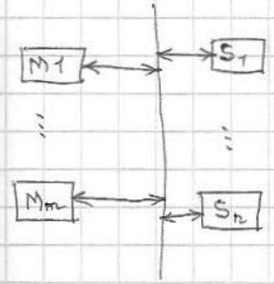
В инт-ных линиях разл-ют виды передачи д-х:

- один ко многим (Multi-drop)



возможность широковещ. передачи от мастера; получает мастер от кого хочет

- Multi-point



I²C 1-Wire SPI Microware JTAG