Pamięć komputerowa (1)

- **podstawowe typy i technologie**
- **hierarchia pamięci**

It would be great to have

unlimited amounts of fast memory…

Trochę historii:

kiedyś, (np. początek lat osiemdziesiątych) w przypadku popularnych komputerów biurowych i domowych sprawa była względnie prosta:

pamięć, z jaką współpracował procesor była podzielona na dwa główne bloki*:

• **ROM** – *Read Only Memory* - tylko do odczytu, nie tracąca danych po wyłączeniu zasilania.

Przechowująca np. podstawowe procedury obsługi urządzeń wejścia/wyjścia (BIOS – *Basic Input Output System*), prosty system operacyjny (np. CP/M), interpreter jakiegoś języka Programowania (np. Basic – był nawet w pierwszych IBM PC) oraz generator znaków.

• **RAM** – *Random Access Memory* – pamięć o „swobodnym dostępie”, czyli taka, której zawartość (dowolne bajty lub dłuższe słowa) można dowolnie zmieniać.

W pamięci RAM przechowywane były (są) zarówno dane jak i kod programu (instrukcje procesora), a po odłączeniu zasilania zawartość takiej pamięci ulega zniszczeniu.

* dodatkowo system mógł być wyposażony w blok pamięci obrazu kontrolera video oraz oczywiście pamięć masową: (taśmy i dyski magnetyczne) – ale do pamięci masowej procesor się z reguły nie odwołuje bezpośrednio…

Trochę historii:

W pierwszych komputerach biurowych (np. do przetwarzania tekstu i prostych arkuszy kalkulacyjnych) szybkość obliczeń nie była głównym priorytetem:

- dysproporcja między szybkością pracy 8/16 bitowego CPU, a **czasem dostępu*** dynamicznej pamięci RAM nie była tak duża jak obecnie, a wstawienie kilku cykli oczekiwania (*wait states*) podczas dostępu do danych nie stanowiło problemu

- IBM w pierwszym PC/XT zastosował okrojoną wersję 16-bitowego procesora i8086 czyli 8088 – z 8-bitową szyną danych - więc dwubajtowe słowa i tak musiały być pobierane bajt po bajcie. Zajmowało dwa razy więcej czasu, za to płyta główna była tańsza w produkcji…

- duża część 8 bitowych komputerów domowych miała jeden blok pamięci RAM, współdzielony z układem generowania grafiki (jego fragment służył również jako pamięć obrazu). Ponieważ układ graficzny musiał pracować non-stop ze stałą prędkością (np. liczbą linii i klatek standardu PAL) – miał wyższy priorytet niż CPU. Na czas odczytu pamięci obrazu przez kontroler wideo procesor był blokowany (w ZX Spectrum zatrzymywano sygnał zegarowy…)

* w uproszczeniu: czas jaki musi upłynąć od podania przez CPU adresu (numeru komórki pamięci) na szynę adresową do pojawienia się odczytanych z pamięci danych (na szynie danych)

Trochę historii:

- W kolejnych latach nastąpił bardzo szybki wzrost wydajności obliczeniowej procesorów (m.in. skrócenie czasu efektywnego cyklu rozkazowego dzięki przetwarzaniu potokowemu).

- Rozwój technologii dynamicznych pamięci RAM – z jakich głównie była budowana podstawowa pamięć operacyjna komputerów przebiega(ł) jednak wolniej.

- Pojawił się problem w postaci znacznej różnicy np. czasami pobrań kolejnych instrukcji (a także ich argumentów) z pamięci, a jej czasem dostępu.

Np. „idealny” procesor potokowy (pełny potok rozkazów, bez dziur i instrukcji warunkowych) pracujący z częstotliwością potoku 500 MHz może pobierać kolejne instrukcje co 2 ns, a czas dostępu do nowego rzędu w dynamicznej pamięci RAM wynosi np. 35 ns.

Z pewną pomocą przychodzi w tym miejscu inny rodzaj pamięci – statyczna pamięć RAM, lecz poza pewnymi szczególnymi przypadkami (np. mikrokontrolery, kilka-kilkaset kilobajtów RAM) nie jest ekonomiczne wykonanie z niej całej pamięci operacyjnej (i video) współczesnego komputera PC (rzędu kilkunastu-kilkudziesięciu GB)

Technologia pamięci komputerowych RAM – Random Access Memory

- Pamięć statyczna (Static RAM)
- Pamięć dynamiczna (Dynamic RAM)

i jej rozwinięcia:

- Synchroniczna dynamiczna (Synchronous Dynamic RAM)

oraz dalsze modyfikacje:

- Double Data Rate - (DDR-SDRAM)

Pamięć Statyczna (Static RAM) – teoria działania

- w **kombinacyjnym** układzie logicznym, (np. omawianym poprzednio sumatorze) stan wyjść zależy tylko od kombinacji aktualnych stanów wejść.

Układ taki jest „**bezpamięciowy**” – stany logiczne, jakie wystąpiły na wejściach „w przeszłości” nie mają żadnego wpływu na aktualny stan wyjść (tj. stan wyjść zależy tylko od aktualnego stanu wejść…)

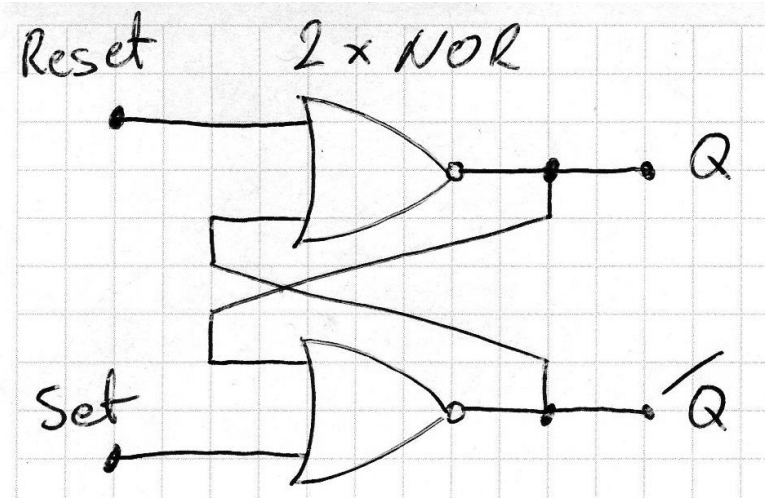
W układzie **sekwencyjnym** istnieje natomiast możliwość „zapamiętania” obecnego stanu. Tym samym kolejny stan układu sekwencyjnego będzie zależał nie tylko od aktualnych stanów wejść, ale również od stanu w jakim układ znajduje się obecnie.

Układami sekwencyjnymi są np.:

- 4-bitowy licznik binarny: po podaniu impulsu na wejście zegarowe zmienia stan czterech wyjść na kolejny np. zliczając w przód: z 0000 na 0001, itd. Osiągnąwszy stan 1111 wraca do 0000 i cała sekwencja się w powtarza.

- procesor: po fazie pobrania instrukcji przechodzi do jej dekodowania, a np. po wykonaniu rozkazu następuje zapis wyniku (po czym cała sekwencja - cykl rozkazowy - się powtarza).

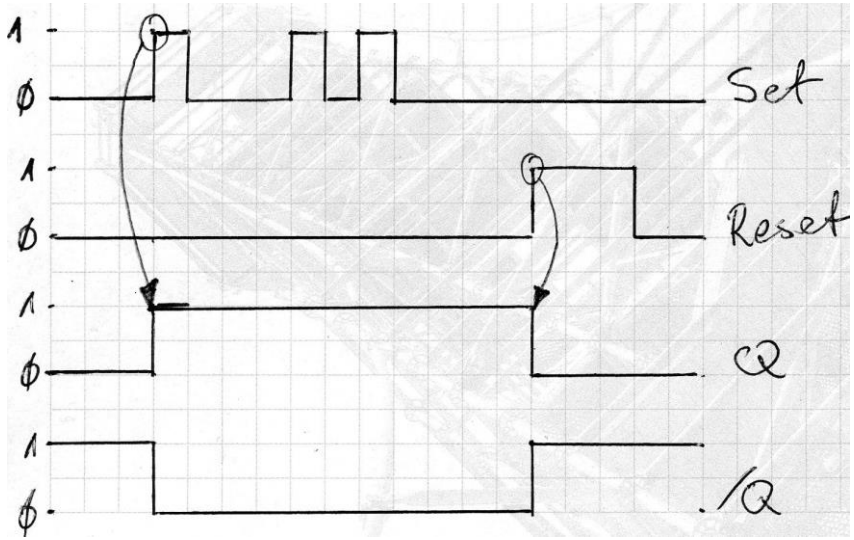
Elementarnym układem sekwencyjnym, pozwalającym zapamiętać jeden bit informacji jest **przerzutnik** asynchroniczny RS (Reset-Set)
można go zbudować z dwóch bramek NOR lub NAND połączonych jak na schemacie:



• Układ posiada dwa wejścia programujące: Set (ustawiające) i Reset (kasujące) oraz dwa wyjścia komplementarne Q i /Q

• Każde z wyjść jest **sprzężone** z wejściem przeciwnej bramki.

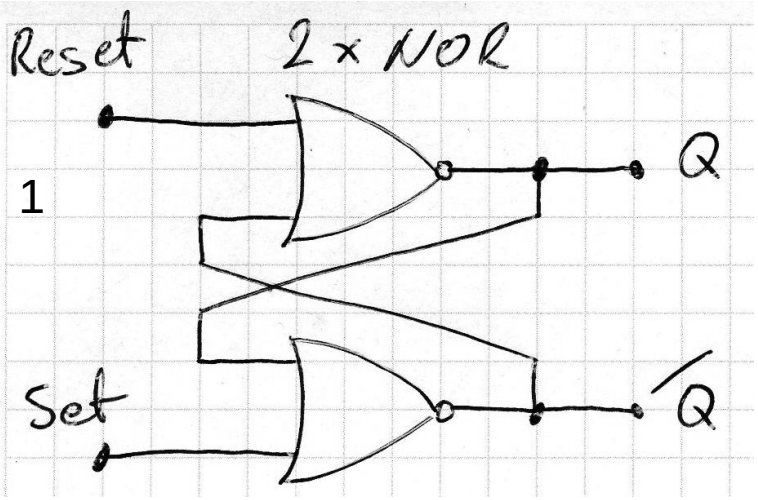
• Tym samym kolejny stan wyjść zależy nie tylko od stanu wejść ale i poprzedniego stanu sygnałów wyjściowych.



• Zmiana stanu przerzutnika następuje w wyniku wzbudzenia impulsem „1” (krótkotrwałym pojawieniem się stanu wysokiego / zbocza narastającego) wejścia:
Set: wtedy Q=1 i /Q=0
Reset: wtedy Q=0 i /Q=1

W stanie „jałowym” (Set=0 i Reset=0) **układ utrzymuje („pamięta”) ostatni wpisany stan.**

1

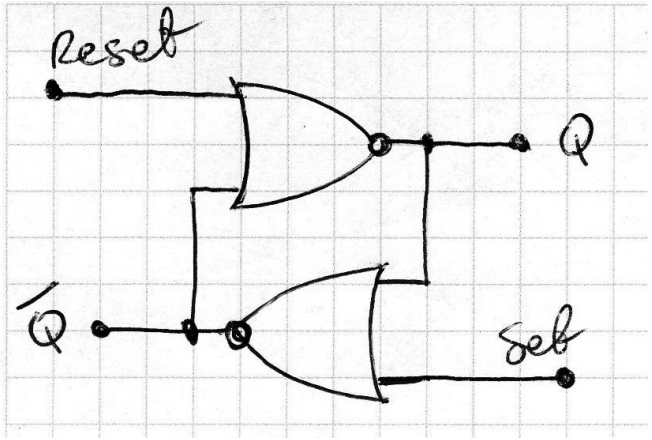


Przerzutnik RS (1) można narysować w inny sposób (2).

Bramki NOR po zwarciu obu wejść zachowują się jak inwerter (NOT).

Cały układ można uprościć (3).

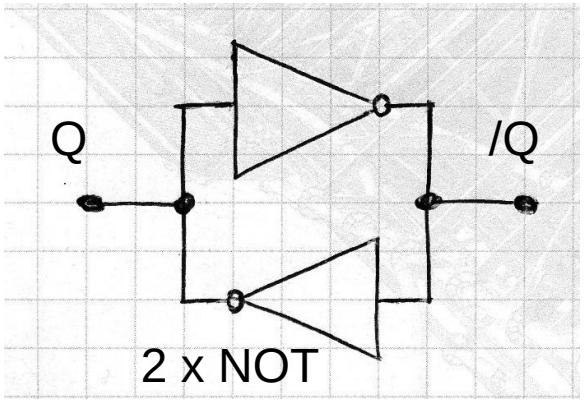
2



• taki układ może się znajdować tylko w **jednym z dwóch stabilnych stanów***: jeśli Q=1 to /Q=0 (lub odwrotnie)

• dany stan jest utrzymywany dopóki układ jest zasilany i nie wymusimy czynnikami zewnętrznymi jego zmiany

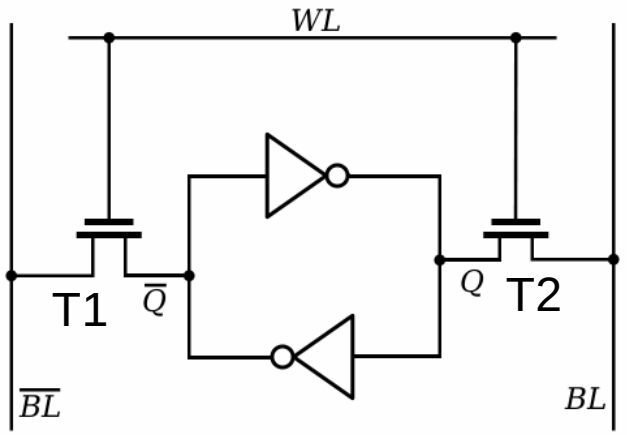
3



•jeśli wymusimy (np. zewnętrznie, poprzez chwilowe zwarcie) stan Q=/Q, to po usunięciu zwarcia układ przejdzie do jednego z dwóch stabilnych stanów. Własność tą można wykorzystać do generowania bitów (i dalej liczb) losowych.

Elementarna komórka (1 bit) pamięci statycznej/rejestru CPU

Aby była możliwość zapisania/odczytu informacji do/z takiej komórki pamięci należy jeszcze dodać tranzystory dostępowe (T1, T2)



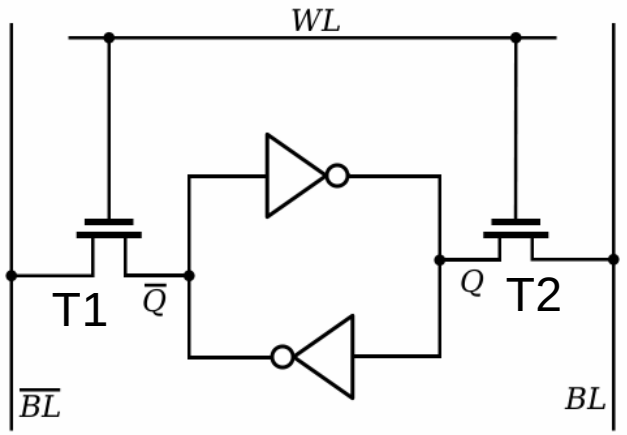
ZAPIS

- Linie BL i /BL (*Bit Line*) ustawiane są w stan odpowiadający zapisywanej informacji

- Linia WL (*Word Line*), wybierająca komórkę pamięci (lub ich zespół: rejestr, bajt, słowo) do której ma się odbyć dostęp jest aktywowana. Tranzystory dostępowe zostają wprowadzone w stan przewodzenia - tym samym stan linii BL i /BL wymusza odpowiednie ustawienie przerzutnika.

- Linia WL jest dezaktywowana, tranzystory T1 i T2 nie przewodzą, przerzutnik zostaje odizolowany od reszty układu i utrzymuje zapisany stan.

Elementarna komórka (1 bit) pamięci statycznej/rejestru CPU



ODCZYT

- Linie BL i /BL służą teraz jako wejścia do następnego bloku (np. bufora odbiorczego).
- Linia WL, wybierająca komórkę pamięci (lub ich zespół: rejestr, bajt, słowo) do której ma się odbyć dostęp, jest na chwilę aktywowana (tranzystory wprowadzone w stan nasycenia).
- Stan linii BL i /BL odpowiada teraz stanom logicznym w węzłach Q i /Q – informacja ta trafia do bufora a następnie na szynę danych.
- Linia WL zostaje dezaktywowana, a przerzutnik odizolowany reszty układu.

Pamięć Statyczna

• jest „szybka” - ma krótki czas dostępu
(w technologii statycznej są wykonane np. rejestry procesora),

• jest „prosta w obsłudze” (cykl zapisu / odczytu) w porównaniu z pamięcią dynamiczną,

• nie wymaga odświeżania zawartości, raz wpisana informacja pozostaje dopóki pamięć jest zasilana (często obniżonym napięciem w stosunku do „roboczego”).

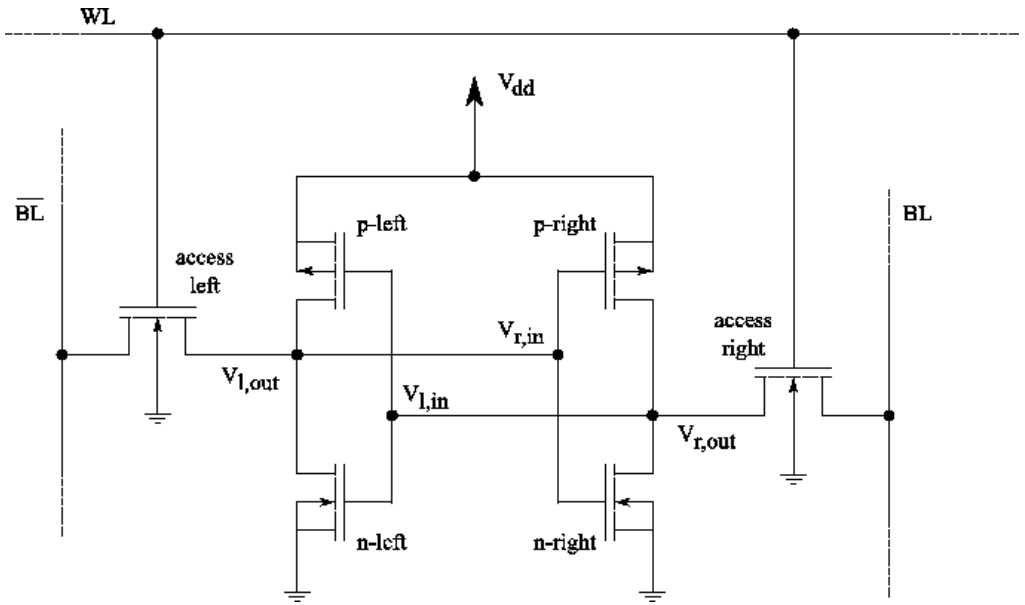
• W przypadku pamięci wykonanej w technologii CMOS i pracy statycznej (raz wpisane dane się nie zmieniają np. konfiguracja komputera) bądź pracy z niską częstotliwością.
(rejestry zegarka: data, godziny, minuty - typowa częstotliwość: 2^15 = 32.768 kHz) pobór prądu jest niewielki*. Zawartość pamięci może być podtrzymywana bateryjnie przez kilka lat.

* podobnie jak w innych układach cyfrowych CMOS, straty energetyczne (np. w postaci wydzielanego ciepła) rosną wraz ze wzrostem częstotliwości przełączania stanów.

Pamięć Statyczna

Wadą pamięci statycznych jest stosunkowo złożona budowa elementarnej komórki - rzutująca na jej większy fizyczny rozmiar.

np. do budowy jednobitowej komórki potrzeba dwóch tranzystorów dostępowych oraz dwóch par komplementarnych (tranzystorów z kanałem N i P) na dwa inwertery (przerzutnik).



Pamięć Statyczna - ciąg dalszy

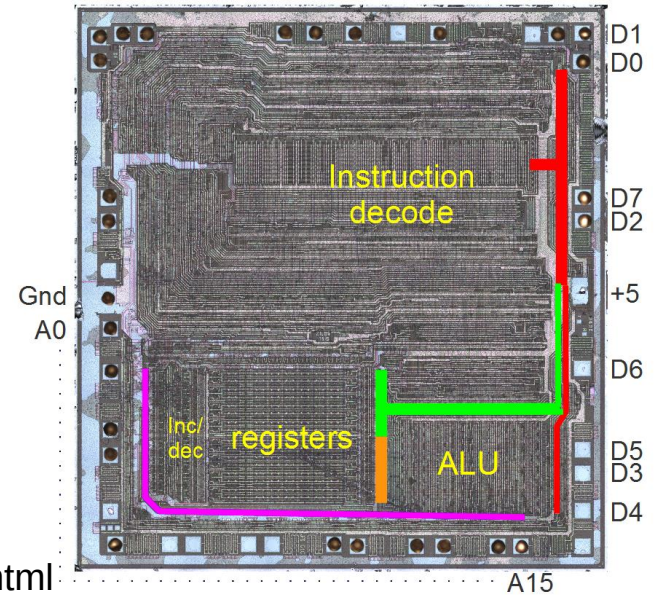
(W uproszczeniu) skomplikowana budowa pamięci statycznej powoduje to, że na danej powierzchni monokrystalicznego krzemu można umieścić mniej komórek niż w przypadku pamięci dynamicznej (zakładając podobne rozmiary np. tranzystorów i szerokości ścieżek).

Mniejsza „gęstość” tej pamięci ma wpływ na wyższą cenę:

np. koszt wyprodukowania 1MB pamięci statycznej jest większy niż koszt 1MB pamięci dynamicznej. Na pamięć o tej samej pojemności trzeba zużyć większą powierzchnię monokrystalicznego, pozbawionego defektów „wafla” krzemowego.

Dodatkowo: im więcej elementów (tranzystorów, złącz itp.) w układzie – łatwiej o defekt któregoś z nich (już na etapie produkcji).

Przykład: miejsce zajmowane przez rejestry w ośmiobitowym procesorze Z80 (2 zestawy po 8 rejestrów 8-bitowych + 5 rej. specjalnych 16 bitowych).



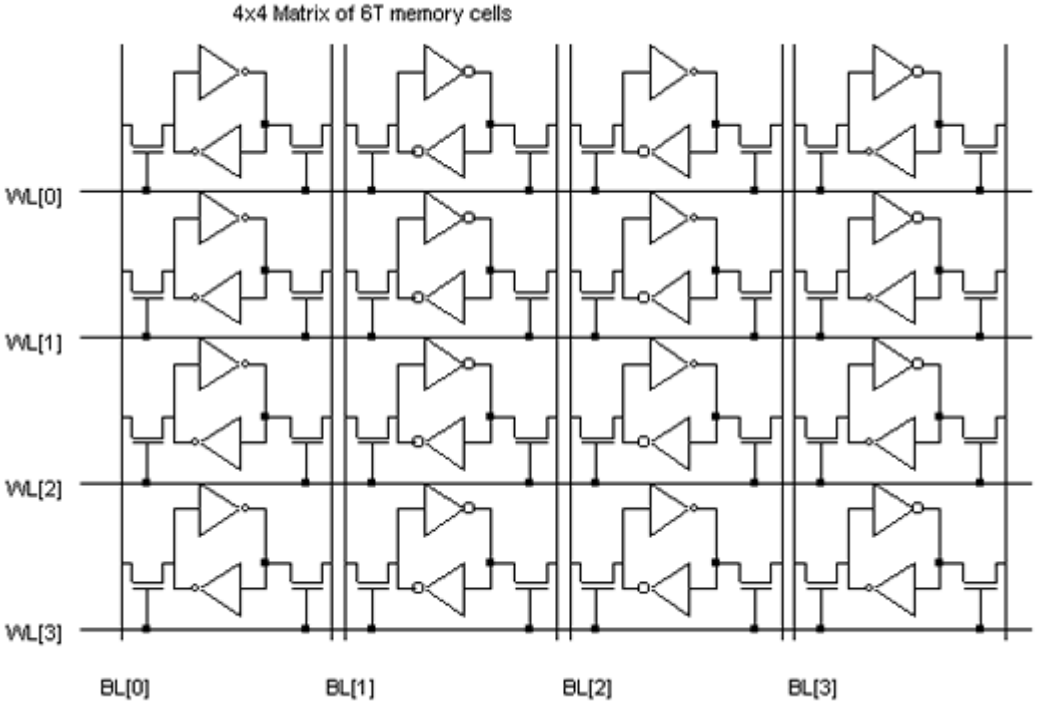
http://www.righto.com/2014/10/how-z80s-registers-are-implemented-down.html

Pamięć Statyczna

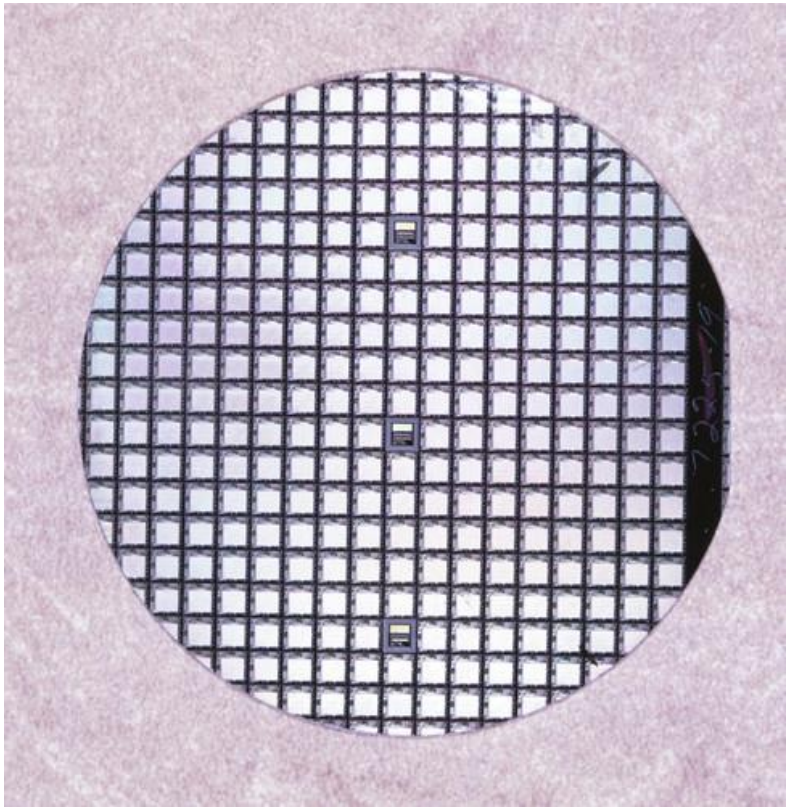
- Jednobitowe komórki pamięci muszą zostać połączone w odpowiedniej długości słowa.

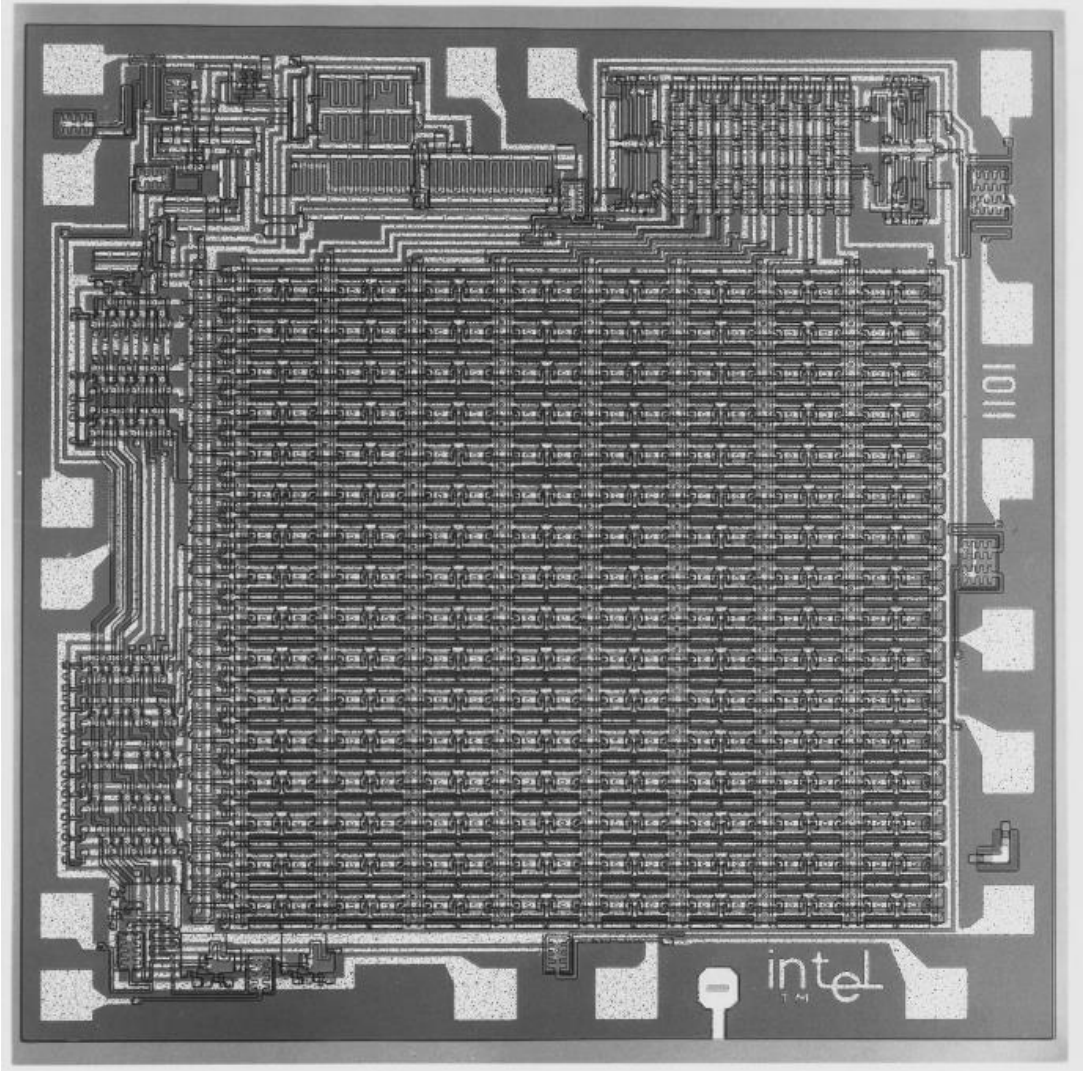
- Typowa pamięć ROM/RAM ma komórki (**niezależnie od ich typu**) ułożone w postaci tablicy składającej się z wielu rzędów i kolumn.

Przykład 16 komórek (statycznych) zorganizowanych w 4 półbajty (nibbles).
Linie WL wybierają dany półbajt, linie BL – zapis/odczyt każdego z 4 bitów półbajtu.



struktura pamięci S-RAM
Intel 1101
(256 x 1 bit,1970 r.)



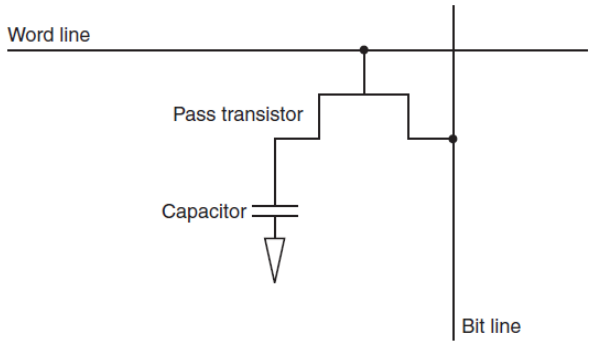


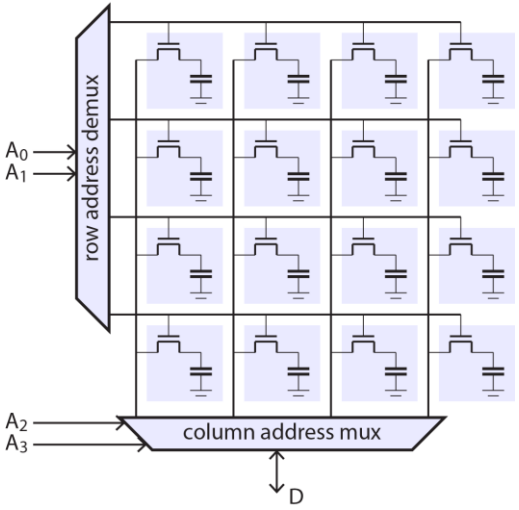
„wafel” 2”

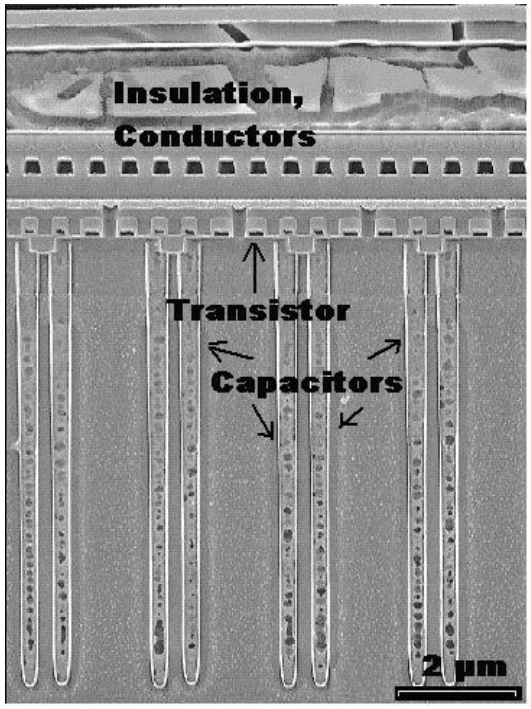
Źródło:
newsroom.intel.com/news/intel-50-intels-1101/#gs.n5eje5
https://calisphere.org/item/ark:/13030/kt2489q185/

Pamięć dynamiczna

- jednobitowa komórka złożona jest z kondensatora i jednego tranzystora dostępowego, dzięki temu jest fizycznie mniejsza, a cała pamięć tańsza niż statyczny odpowiednik.







- zapisany w pamięci stan logiczny dopowiada ładunkowi zgromadzonemu w kondensatorze („rozładowany” - „naładowany”…)
- kondensator ma upływność, więc zgromadzony ładunek nie utrzymuje się w nim w nieskończoność…
- zawartość pamięci musi być zatem odświeżana co pewien czas (*refresh*) w uproszczeniu: odczytana i zapisana ponownie na swoim miejscu
- czas dostępu jest dłuższy, a obsługa bardziej skomplikowana (niż w statycznej)

Pamięć dynamiczna

Obsługa pamięci dynamicznej:

- Adres dzieli się na dwie części - rzędu (*row address*) i kolumny (*column address*).
- Jako pierwszy podawany jest adres rzędu.
- Następuje otwarcie (dostęp do) rzędu: napięcia panujące na kondensatorach komórek w otwieranym rzędzie są próbkowane, wzmacniane a odpowiadające jej wartościom stany logiczne zapisywane w buforze.
    - Otwierany zawsze jest rząd (w uproszczeniu: tyle elementów – ile kolumn…), a nie pojedyncze bity (bajty). Ten etap znacznie wydłuża całkowity czas dostępu.
- Dane z bufora mogą już być swobodnie i szybko odczytane (wszystkie kolumny, bądź tylko niektóre – wybierane adresem kolumny) i przesłane np. do procesora.
- W tym samym czasie (gdy dane są już gotowe do odczytu) odbywa się odświeżenie (refresh) zawartości odczytanego rzędu – kondensatory komórek są ponownie ładowane do wartości napięć odpowiadającym zapamiętanym w buforze stanom logicznym.

Synchroniczna Dynamiczna RAM (SDRAM)

- Przesyłanie rozkazów sterujących i danych z / do pamięci odbywa się synchronicznie – w takt sygnału zegarowego.

- Dane mogą być przesyłane na obu zboczach (rosnącym/malejącym) sygnału zegarowego (*DDR - Double Data Rate*).

- Pamięć podzielona jest na tzw. banki (np. cztery) – a każdy bank zawiera w sobie tablicę komórek zorganizowaną w rzędy i kolumny.

- Możliwe jest otwieranie wielu rzędów w wielu bankach, bądź praca z przeplotem (*interleaved/pipelined access* – o tym dalej).

- Oddzielne magistrale: adresowa, danych i sterująca.

- *Burst mode* - możliwość transferu danych seriami (o np. programowanej długości) bez konieczności przesyłania adresów kolejnych kolumn.

Synchronous Dynamic RAM (SDRAM)

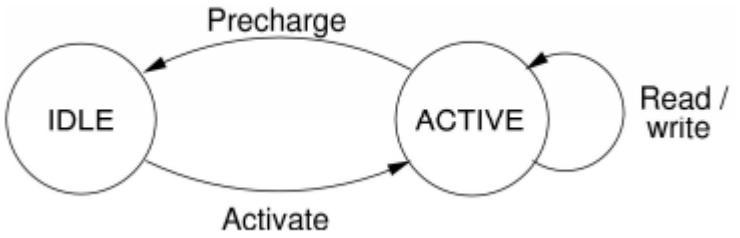
Commands:

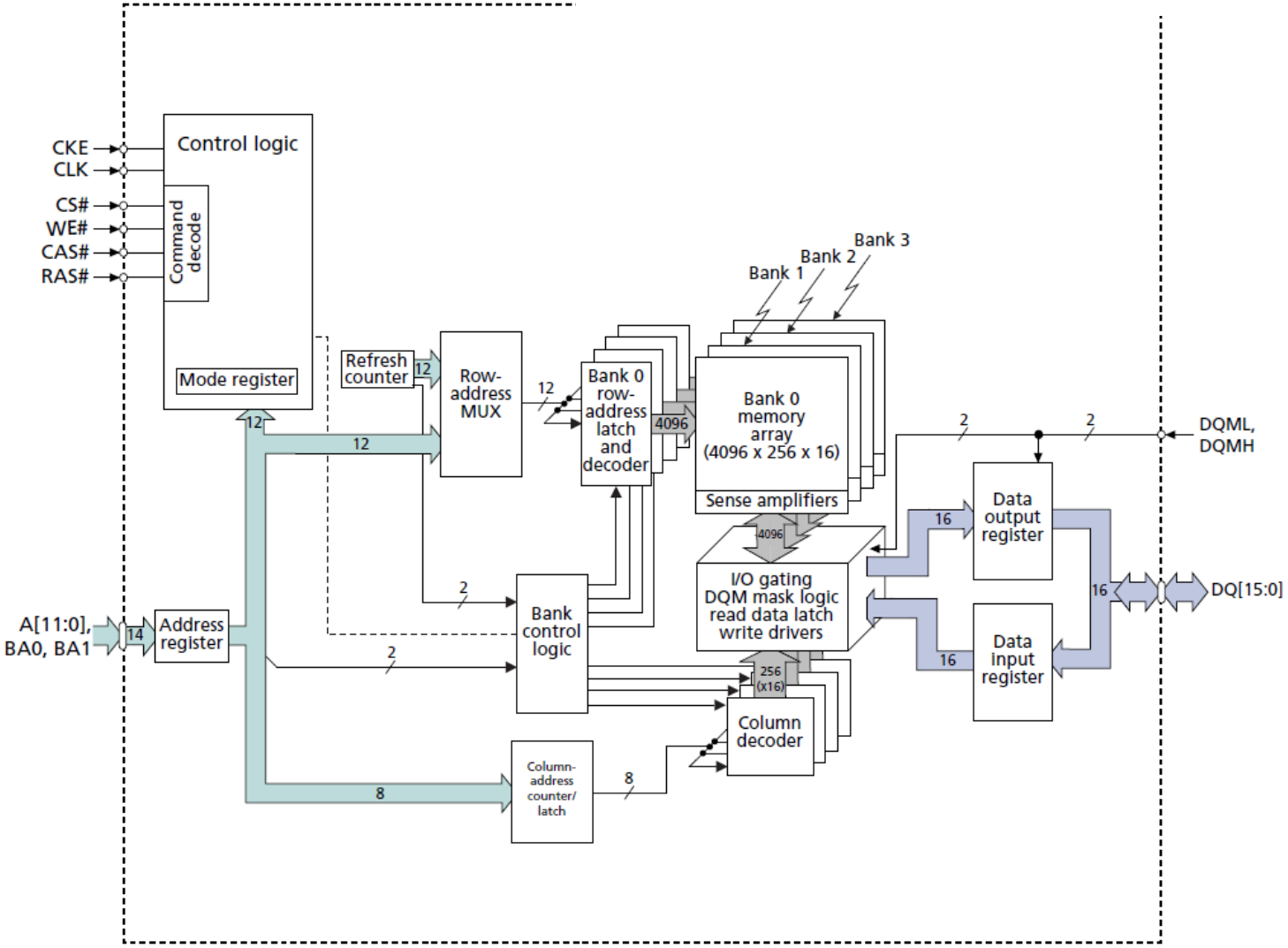
ACTIVATE
- open the row -
data are transferred
from selected
row of cells
to data latch
(buffer)

READ/WRITE
Burst data transfer

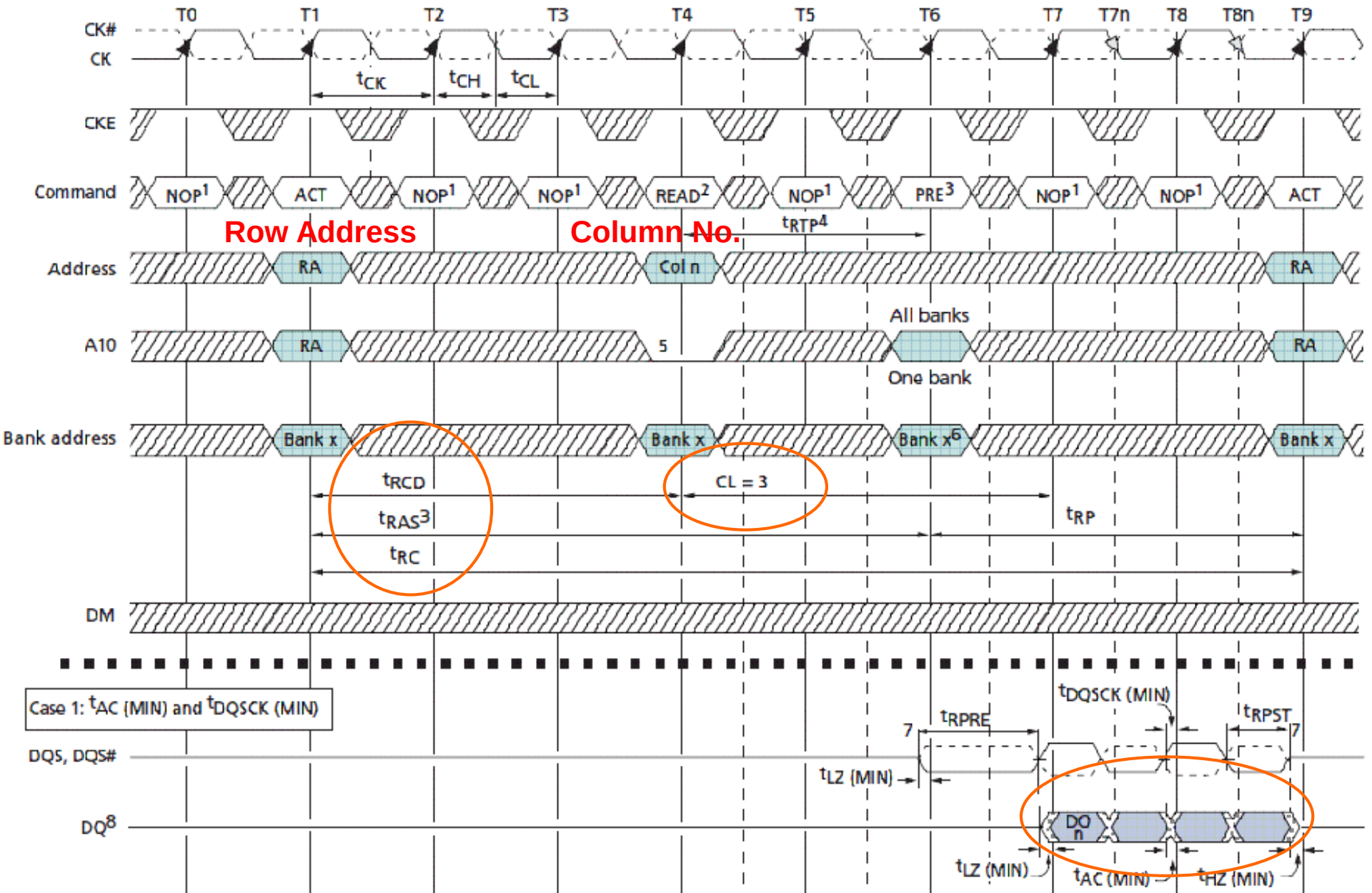
PRECHARGE
-close the row-
write data back
(latch -> memory),
reset sense amps,
precharge bit lines,
(and eventually
select next bank)

REFRESH
e.g. 64ms/all rows





Double Data Rate Synchronous Dynamic RAM (DDR-SDRAM)



Synchronous Dynamic RAM (SDRAM)

Dostęp z przeplotem (*interleaved / pipelined access*)

- Pamięć podzielona jest na banki (mogą to też być całe moduły pamięci).
- Kiedy po aktywacji / otwarciu rzędu w banku *n* uzyskujemy dostęp do danych, równocześnie otwiera się następny rząd w banku *n+1.*
- Pomysł podobny do przetwarzania potokowego instrukcji przez CPU.

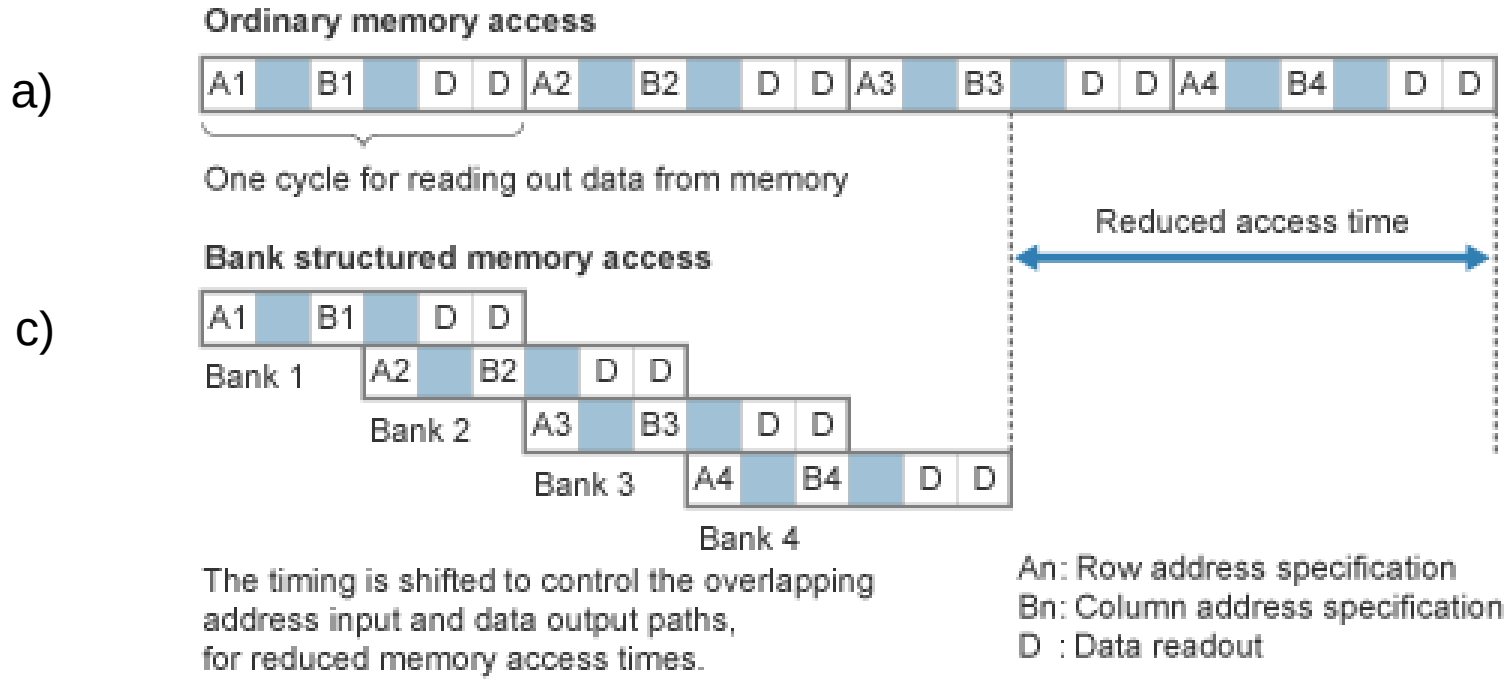
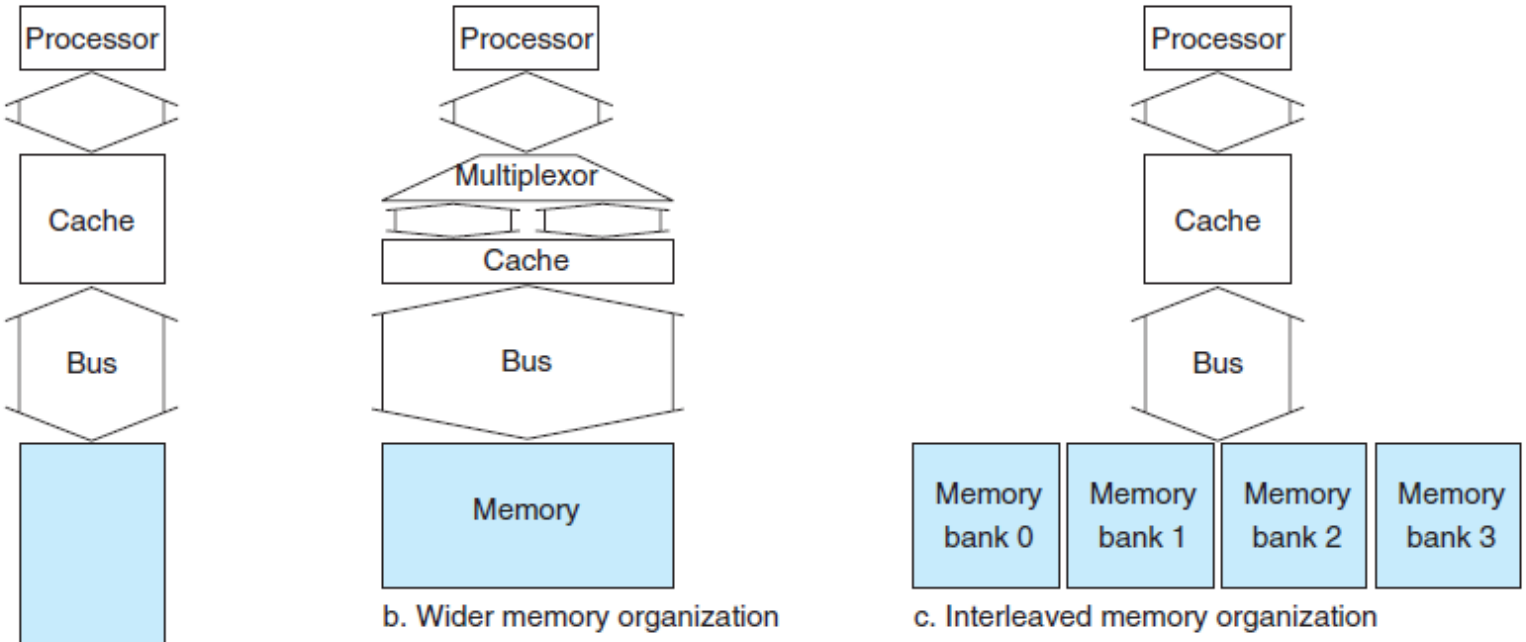


Fig. 2: Comparison of ordinary memory access and interleaved memory access

Typowe rozwiązania połączenia CPU - pamięć



64 bitowy CPU i odpowiednio szersza 128/192/256 bitowa szyna pamięci pozwala pobrać odpowiednio więcej danych podczas jednego dostępu (otwarcia rzędu). Konstrukcja szybka, ale droższa w budowie. np. niektóre x86-64, procesory graficzne (GPU) i High Bandwidth Memory Hybrid Memory Cube, ale również 32 bitowy ARM Cortex M4 z 128 bitowa pamięć flash (wewnątrz mikrokontrolera)

Dostęp z przeplotem: kompromis pozwalający zwiększyć przepustowość, jednocześnie umożliwiając stosowanie tańszej pamięci z węższą szyną danych

dostęp sekwencyjny,
np. 32 bitowy procesor i tania pamięć
z wąską, 32 bitową szyną danych np. i80386DX

Schematy a i c odnoszą się również do rysunków a i c z poprzedniego slajdu.

type vs. speed vs. cost vs. capacity

Memory technology	Typical access time	$ per GIB in 2012
SRAM semiconductor memory	0.5–2.5 ns	$500–$1000
DRAM semiconductor memory	50–70 ns	$10–$20
Flash semiconductor memory	5,000–50,000 ns	$0.75–$1.00
Magnetic disk	5,000,000–20,000,000 ns	$0.05–$0.10

Year introduced	Chip size	$ per GiB	Total access time to a new row/column	Average column access time to existing row
1980	64 Kibibit	$1,500,000	250 ns	150 ns
1983	256 Kibibit	$500,000	185 ns	100 ns
1985	1 Mebibit	$200,000	135 ns	40 ns
1989	4 Mebibit	$50,000	110 ns	40 ns
1992	16 Mebibit	$15,000	90 ns	30 ns
1996	64 Mebibit	$10,000	60 ns	12 ns
1998	128 Mebibit	$4,000	60 ns	10 ns
2000	256 Mebibit	$1,000	55 ns	7 ns
2004	512 Mebibit	$250	50 ns	5 ns
2007	1 Gibibit	$50	45 ns	1.25 ns
2010	2 Gibibit	$30	40 ns	1 ns
2012	4 Gibibit	$1	35 ns	0.8 ns

Podsumowanie

- Prosta w obsłudze, szybka pamięć statyczna ma czas dostępu (i zapisu danych) wystarczająco krótki by umożliwić bezpośrednią pracę z jednostkami wykonawczymi współczesnego procesora.

- Rozwiązanie takie znajduje zastosowanie w przypadku rejestrów procesora, ew. niewielkiej pamięci RAM mikrokontrolerów.

- Jest ono jednak nieekonomiczne z punktu widzenia budowy całej pamięci operacyjnej współczesnego komputera (np. kilkanaście-kilkadziesiąt GB).

- Możliwa jest masowa produkcja tanich układów pamięci dynamicznej o bardzo dużej pojemności (i to jest praktycznie jedyna zaleta).

- Niestety, czas dostępu do nowego rzędu komórek przekreśla bezpośrednie (wydajnościowo efektywne) połączenie pamięci dynamicznej ze współczesnym procesorem.

Powszechnie przyjęte **rozwiązanie tego problemu** wymaga (przynajmniej pobieżnej…) analizy elementów typowego programu komputerowego.

Przykład – wyszukaj min i max element tablicy

```
table:    .long 4,2,8,6,4,1,9,4          #vector of eight unsigned integers
string:   .ascii ‘min=%u, max=%u\n\0’

min_max:
  mov     $7,%esi                        #number of elements minus one
  mov     $0xFFFFFFFF, %eax              #minimum in eax
  xor     %ebx,%ebx                      #maximum in ebx

min_max_loop:
  mov     table(,%esi,4),%edx
  cmp     %eax,%edx
  cmovb   %edx,%eax                      #conditional move if below
  cmp     %ebx,%edx
  cmova   %edx,%ebx                      #conditional move if above
  dec     %esi
  jns     min_max_loop

mov       $string,%rdi
mov       %eax,%esi
mov       %ebx,%edx
xor       %eax,%eax
call      printf@plt                     # call dynamically-loaded function
                                         # an indirect and „far” jump
ret                                      # return - implied memory access (stack operation)
                                         # and jump
```

Lokalność czasowa i przestrzenna

Okazuje się, że większość poprawnie napisanych* programów cechuje:

• **temporal locality** – lokalność czasowa

ostatnio pobrane z pamięci i wykonane instrukcje (i przetwarzane dane!) mogą być za chwilę znowu potrzebne:

- bloki kodu powtarzane w pętli,
- stałe niezbędne do obliczeń: PI, e, stała Plancka itp…

• **spatial locality** – lokalność w przestrzeni adresowej (przestrzenna)

- jeśli procesor wykonuje *n*-tą instrukcję programu, za chwilę prawdopodobnie będzie musiał pobrać *n*+1, z sąsiednich komórek pamięci (o ile nie wystąpi skok!)

- jeśli przetwarzany jest *n*-ty element wektora/macierzy jest bardzo prawdopodobnym, że za chwilę będzie potrzebny kolejny…

(proszę pamiętać, że wiele danych jest przetwarzanych w postaci wektorów i macierzy w tym: sygnały i strumienie audio i video, grafika 3D)

*Lokalności w kodzie i w dostępie do danych można poprawić, bądź zaburzyć

np. w przypadku:

```
cmp     %eax,%edx       #porównaj i ustaw flagi
cmovb   %edx,%eax       #skopiuj warunkowo, jeśli układ flag odpowiada warunkowi „below”
cmp     %ebx,%edx       #czyli carry=1
cmova   %edx,%ebx       #skopiuj warunkowo, jeśli układ flag odpowiada warunkowi „above”
```

instrukcje pobierane są „liniowo”, jedna po drugiej, a jeśli warunek nie jest spełniony CPU ignoruje wykonanie rozkazu *cmov*. Jest to ułatwienie dla procesora potokowego – nie musi spekulować/przewidywać czy skok będzie wykonany, oraz nie musi obliczać adresu skoku - z jakiego pobrać kolejne instrukcje.

W klasycznej postaci („*if*” ze skokami warunkowymi) – lokalność przestrzenna pogorszy się (tutaj widoczne przeskoki tylko o jedną instrukcję):

```
cmp     %eax,%edx
jae ae
mov     %edx,%eax
ae:
cmp     %ebx,%edx
jbe end
mov     %edx,%ebx
end:
```

Hierarchia pamięci komputera

Istnienie lokalności czasowych i przestrzennych w programach komputerowych wykorzystane zostało przy projektowaniu pamięci systemów mikroprocesorowych.

- między małą liczbą szybkich rejestrów procesora, a dużym obszarem wolnej pamięci RAM wprowadzono niewielki blok (lub obecnie nawet kilka) szybkiej **pamięci podręcznej** (cache), przechowującej **ostatnio pobrane** instrukcje, dane oraz ich **najbliższe „sąsiedztwo”**.

W praktyce mamy do czynienia z **hierarchią pamięci** w komputerze: strukturą (sprzętową) składającą się z wielu poziomów – bloków pamięci o różnej pojemności i czasie dostępu (plus połączeń między nimi i układ sterowania).

Generalnie im „bliżej” procesora tym pamięć jest „szybsza” (droższa…), ale jest jej „mniej”.

Ogólna zasada działania jest stosunkowo prosta…

Procesor chcąc pobrać dane/instrukcje „sięga” w pierwszej kolejności do pamięci najbliższej-najszybszej, jeśli ich tam nie ma – wymagane elementy są poszukiwane w dalszych-wolniejszych poziomach i „ściągane” do pamięci o krótszym czasie dostępu, a następnie do procesora. Sprawę zapisu danych na razie pomijamy.

Hierarchia pamięci komputera

Pamięć podręczna - cache - statyczna
L1 - oddzielna dla danych i instrukcji
L2 - każdy rdzeń ma swoją
L3 - przeważnie współdzielona między rdzeniami

główna pamięć operacyjna (dynamiczna, obecnie SDRAM)

pamięć masowa – najwolniejsza najmniejszy koszt 1GB (1TB)
(dyski, pamięci Flash, streamery)
+ ew. plik wymiany pamięci wirtualnej