Логичко пројектовање

Предавање 10/10

Увод у пројектовање рачунарских система

Увод

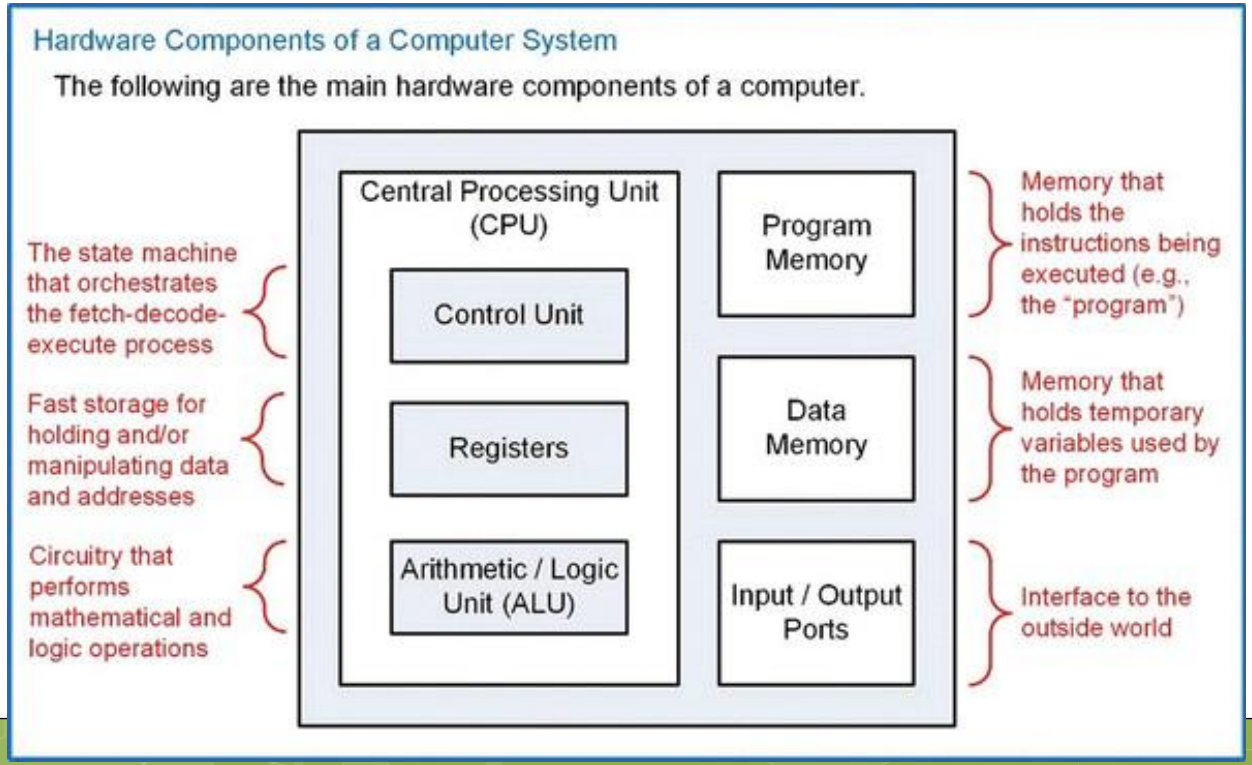
- Један од најчешћих дигиталних система који се данас користи је рачунар.
- Рачунар извршава задатке кроз архитектуру која користи хардвер и софтвер.
- Хардвер у рачунару се састоји од многих елемената који су до сада представљени. То укључује регистре, аритметичка и логичка кола, коначне аутомате и меморију.
- Оно што рачунар чини тако корисним је што је хардвер дизајниран да оствари унапред одређени скуп **инструкција**.
- Ове инструкције су релативно једноставне, као што је на пример пренос података између меморије и регистра или извођење аритметичке операције између два броја.
- Инстарукције се састоје од бинарних кодова који се чувају у меморији и представљају секвенцу операција које ће хардвер извршити да би извршио неки задатак.
- Ова секвенца инструкција назива се **компјутерски програм**.
- Оно шта чини ову архитектуру корисном је да се већ постојећи хардвер може програмирати да обавља скоро неограничен број задатака једноставним дефинисањем секвенце инструкција које треба извршити.
- Процес пројектовања секвенце инструкција или програм се назива развој софтвера или софтверски инжењеринг.

Историја рачунарских машина

- Идеја о рачунарској машини опште намене датира још од деветнаестог века.
- Прве рачунарске машине су имплементиране са механичким системом и били су типично аналогни по природи.
- Како је технологија напредовала, компјутерски хардвер је еволуирао од електромеханичких прекидача до вакуумске цеви и на крају до интегрисаних кола.
- Ове новије технологије омогућиле су прекидачка кола и обезбедиле могућност изградње бинарних рачунара. Данашњи рачунари су направљени искључиво од полупроводничких материјала и технологије интегрисаних кола.
- Термин микрорачунар се користи за описивање рачунара који има хардвер за процесирање имплементиран са интегрисаним колима. Скоро сви савремени рачунари су бинарни. Бинарни рачунари су дизајнирани да раде са фиксним скупом битова. На пример, 8-битни рачунар ради операције са 8 бита одједном. То значи да помера податке између регистара и меморију и изводи аритметичке и логичке операције у групама од 8 бита.
- Ова лекција покрива основе једноставног рачунарског система и представља дизајн 8-битног система за илустрацију детаља извршења инструкција.
- Циљ је разумевање основних принципа рачунарских система.

Рачунарски хардвер

- Рачунарски хардвер се односи на све физичке компоненте унутар система, као што су меморија, уређаји, регистри и коначни аутомати.



Програмска меморија

- Инструкције које извршава рачунар чувају се у програмској меморији.
- Програмска меморија се третира као меморија само за читање током извршавања инструкција како би се спречило да рачунар обрише инструкције.
- Неки компјутерски системи имплементирају програмску меморију помоћу праве ROM меморије (MROM или PROM), док ће други користити EEPROM из којег се може читати током нормалног рада, али се може уписати само помоћу наменске процедуре за упис.
- Програми се обично чувају у постојаној меморији тако да их рачунарски систем не изгуби када се искључи струја.
- Савремени рачунари често копирају програм из постојане меморије (нпр. хард диск) у привремену меморију након покретања како би се убрзало извршење инструкција. У овом случају, потребно је обратити пажњу да програм не препише сам себе.

Меморија за податке

- Рачунари такође захтевају меморију за податке у коју се може уписивати и читати током нормалног рада рачунара.
- Ова меморија се користи за чување привремених променљивих које се креирају софтверским програмом.
- Она проширује способност рачунарских система омогућавајући стварање велике количине информација које програм чува.
- Осим тога, могу се извршити израчунавања која су већа од ширине рачунарског система чувајући привремене резултате израчунавања (на пример израчунавање 128-битног сабирања на 32-битном рачунару).
- Меморија за податке се имплементира са R/W меморијом, најчешће SRAM или DRAM.

Улазно-излазни портови

- Термин порт се користи да опише механизам за добијање информација од спољашњег света у рачунар или из њега.
- Портови могу бити улазни, излазни или двосмерни. И/О портови могу бити дизајнирани да преносе информације серијски или паралелно.

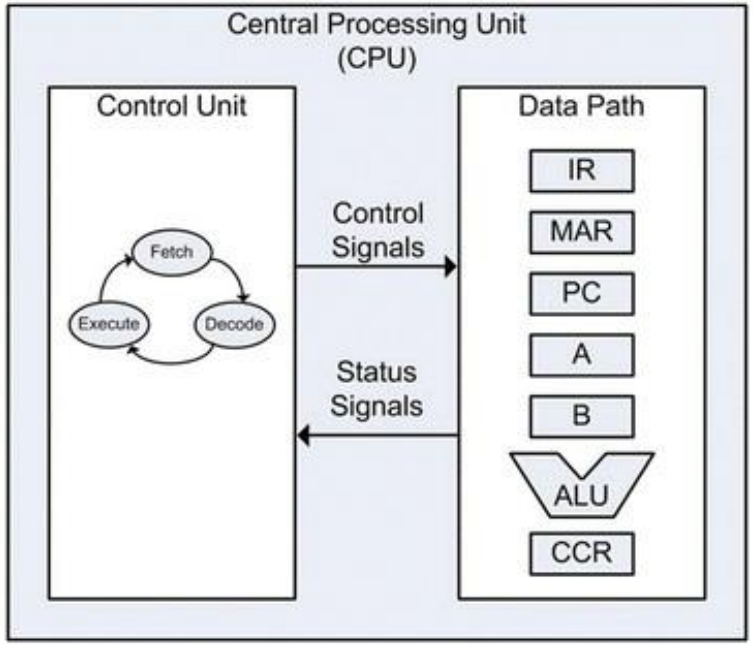
CPU

- Централна процесорска јединица (CPU) се сматра мозгом рачунара.
- CPU управља читањем инструкција из меморије, декодира их да би разумео која врста инструкције се извршава и извршава неопходне кораке за комплетирање инструкције.
- CPU такође садржи скуп регистара који се користе за складиштење података опште намене, оперативне информације и статус система.
- Коначно, CPU садржи кола за обављање аритметичких и логичких операција над подацима.

Организација CPU

Typical CPU Organization

A CPU is functionally organized into a control unit and a data path. The control unit contains the FSM to orchestrate the fetch-decode-execute process. The registers and ALU are grouped into a unit called the data path. The control unit sends control signals to the data path to move and manipulate data. The control unit uses status signals from the data path to decide which states to traverse in its FSM.



Контролна јединица

- Управљачка јединица је коначни аутомат који контролише рад рачунара.
- Овај коначни аутомат има стања која врше преузимање инструкције (тј. читање из програмске меморије), декодирање инструкције и извршавање одговарајућих корака за остваривање инструкције.
- Овај процес је познат као прибављање, декодирање, извршење и понавља се сваки пут када се инструкција извршава у CPU.
- Док коначни аутомат контролне јединице пролази кроз своја стања, он управља контролним сигналима који врше пренос и манипулацију са подацима у циљу постизања жељене функционалности инструкције.

Ток података : Регистри

- CPU групише своје регистре и ALU у подсистем који се зове ток података.
- Ток података се односи на брзо складиштење и манипулацију подацима унутар CPU-а.
- Све ове операције иницира и њима управља коначни аутомат контролне јединице.
- CPU садржи низ регистара који су неопходни за извршавање инструкција и информације о статусу система.
- Рачунари имају следеће регистре у њиховом CPU-у:
- **Регистар инструкција (IR)** — Регистар инструкција смешта тренутни бинарни код инструкције која се извршава. Овај код се чита из програмске меморије као први део извршења инструкција. IR користи контролну јединицу како би одлучио која стања у свом коначном аутомату да пређе како би се извршила инструкција.
- **Регистар адреса меморије (MAR)** — MAR се користи за задржавање тренутне адресе која се користи за приступ меморији. MAR се може учитати са адресама у циљу преузимања инструкција из програмске меморије или са адресе за приступ меморији за податке и/или I/O портовима.
- **Програмски бројач (PC)** — Програмски бројач садржи адресу тренутне инструкције која се извршава у програмској меморији. Програмски бројач ће се повећавати узастопно када се инструкцијњ читају из програмске меморије све док се наменска инструкција не користи за постављање на нову локацију.
- **Регистри опште намене** — Ови регистри су доступни за привремено складиштење од стране програма. Постоје инструкције за премештање информација из меморије у ове регистре и за премештање информација из ових регистара у меморију. Постоје и инструкције за извршење аритметиких и логичких операција над подацима који се налазе у овим регистрима.
- **Регистар кодова стања (CCR)** — CCR садржи статусне флегове који пружају информације о аритметичким и логичким операцијама које се изводе у CPU. Најчешћи флегови су негативни (N), нула (Z), прекорачење двоструког комплемента (V) и пренос (C). Овај регистар такође може да садржи флегове који указују на статус рачунара, на пример да ли има прекиде или ако је рачунар стављен у режим ниске потрошње енергије.

Ток података : ALU

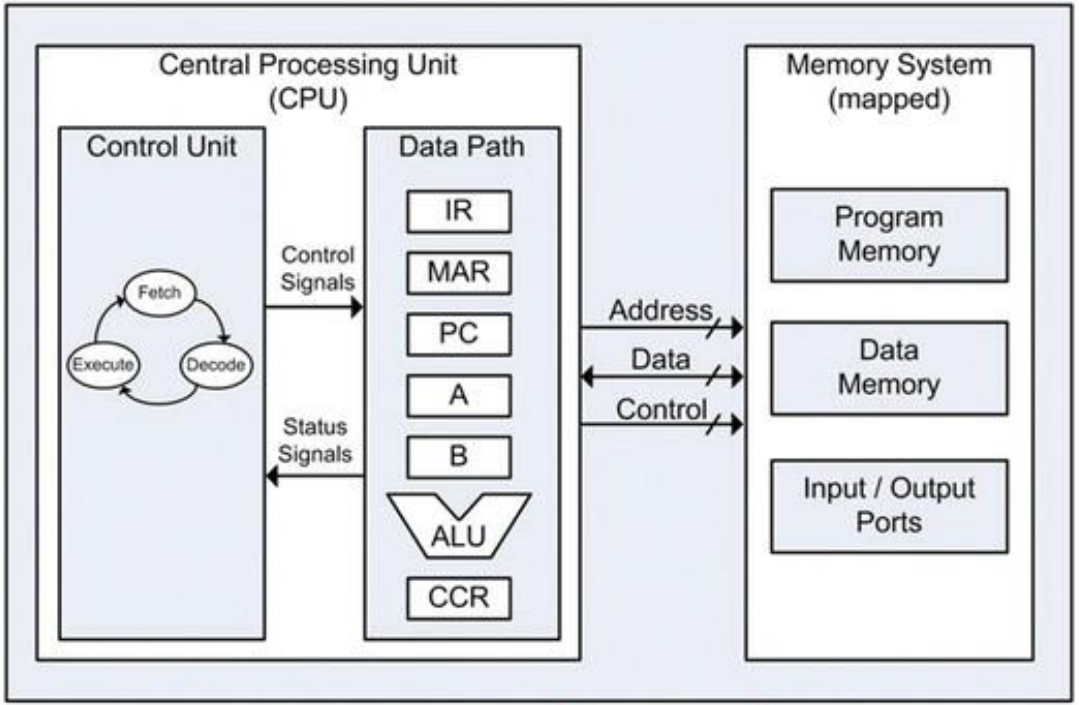
- Аритметичко-логичка јединица (ALU) је систем који обавља све математичке (тј. сабирање, одузимање, множење и дељење) и логичке операције (тј. and, or, not, shift).
- Овај систем ради на подацима који се чувају у CPU-у регистрима.
- ALU има јединствени симбол повезан са њим да би се разликовао од других функционалних јединица у CPU-у.

Систем са мапирањем мем.

-

Computer Hardware in a Memory Mapped Configuration

In a memory mapped system, unique addresses are assigned for all locations in program and data memory in addition to each I/O port. In this way the CPU can access everything using just an address.

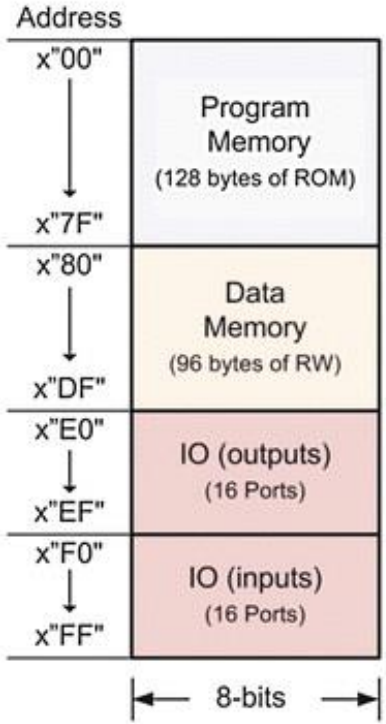


A bus system is used to move information between the memory system and the CPU.

Пример мапирања мем.

Example: Memory Map for a 256x8 Memory System

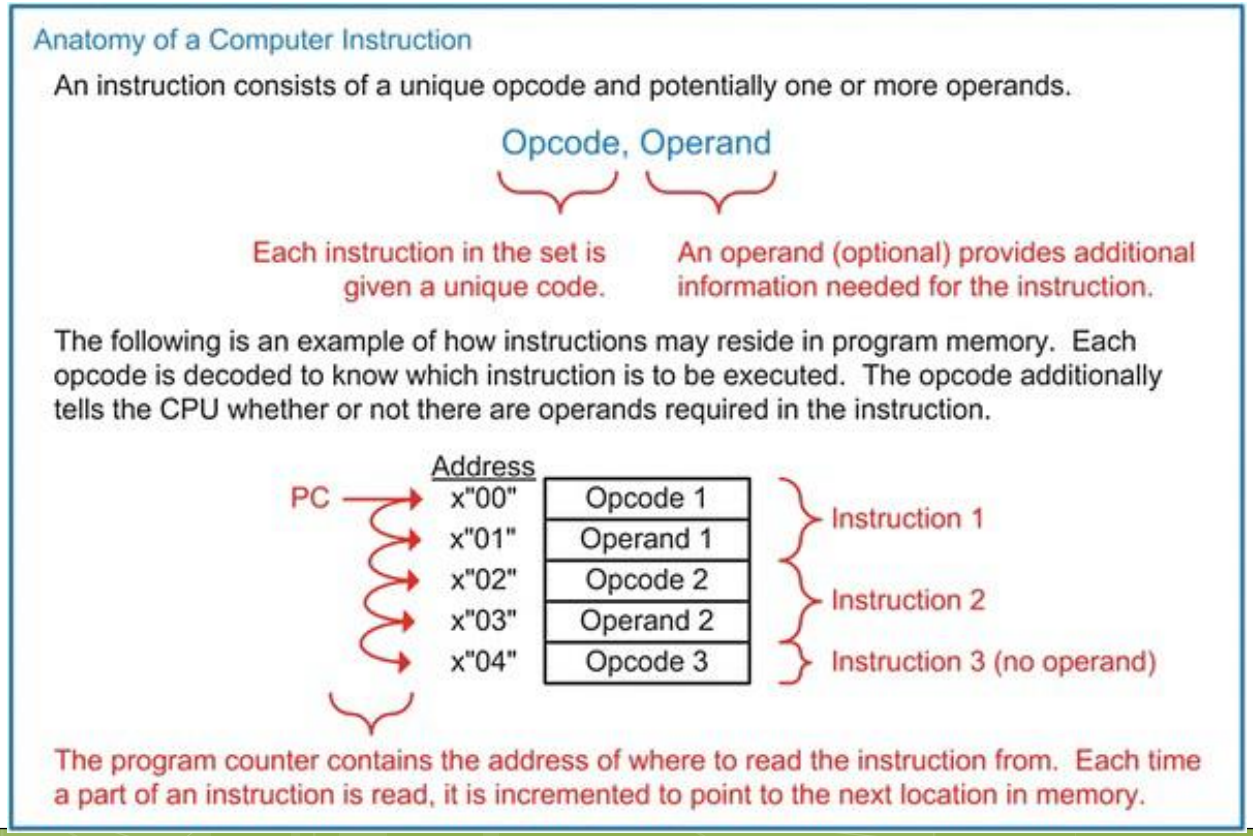
The following is a memory map for an example 8-bit computer system.



Рачунсарски софтвер

- Рачунарски софтвер се односи на инструкције које рачунар може да изврши и како су оне дизајниране да остваре различите задатке.
- Специфична група инструкција које рачунар може да изврши је познат као његов **скуп инструкција**.
- Скуп инструкција рачунара треба прво да се дефинише пре имплементације рачунарског хардвера.
- Неки рачунарски системи имају веома мали број инструкција у циљу смањења физичке величине кола потребних у CPU-у. Ово омогућава CPU-у да изврши инструкције веома брзо,али захтева велики број операција за остварење датог задатка. Овај оархитектонски приступ се назива рачунар са смањеним скупом инструкција (**RISC**).
- Алтернатива овом приступу је да се направи скуп инструкција са великим бројем наменских инструкција које могу да остваре дати задатак са мање CPU-аоперација. Недостатак овог приступа је физичка величина CPU-а, мора да буде већи да би могао да прими разне инструкције. Овај архитектонски приступ се назива рачунар са сложеним скупом инструкција (**CISC**).

Opcode-ови и Operand-и



Режим адресирања за Operand-е

- Режим адресирања описује начин на који је операнд инструкције коришћен.
- Док савремени рачунарски системи могу да садрже бројне начине адресирања са различитим сложеностима, фокусираћемо се само на подскуп основног адресирања. Ови начини су непосредни, директни, инхерентни и индексирани.
- **Непосредно адресирање (IMM)** је када је операнд инструкције информација која се користити од стране инструкције. На пример, ако је постојала инструкција да се стави константа у регистар унутар CPU-а користећи непосредно адресирање, операнд би била константа. Када CPU чита операнд, он једноставно убацује садржај у CPU регистар и инструкција је завршена.
- **Директно адресирање (DIR)** Директно адресирање је када операнд инструкције садржи адресу где се налазе информације које треба користити. На пример, ако инструкција треба да стави константу у регистар унутар CPU-а користећи директно адресирање, операнд би садржао адресу на којој се константа налази у меморији. Када CPU чита операнд, он ставља ову вредност на магистралу адреса и врши додатно читање да преузме садржај који се налази на тој адреси. Прочитана вредност се затим ставља у CPU регистар и инструкција је комплетна.

Режим адресирања за Operand-е (наставак)

- **Инхерентно адресирање (INH)** се односи на инструкцију која не захтева операнд јер сам opcode садржи све потребне информације за инструкцију коју треба извршити. Ова врста адресирања се користи за инструкције које врше манипулацију над подацима који се држе у регистрима CPU-а без потребе за приступом меморијском систему. На пример, ако је постојала инструкција за повећање садржај регистра (A), онда када CPU прочита opcode, он зна све шта треба да би извршио задатак. CPU једноставно потврђује низ контролних сигнала у циљу повећања садржаја. За овај задатак није потребан операнд. Уместо тога, локација регистра којим се манипулише (тј. A) је инхерентна унутар opcode.
- **Индексирано адресирање (IND)** се односи за инструкције које приступају информацијама на адреси у меморији, али адреса којој треба приступити се чува у другом CPU регистру. У овој врсти адресирања, операнд инструкције се користи као офсет који се може применити на адресу која се налази у CPU регистру. На пример, нека постоји инструкција у коју треба константу уписати у регистар (A) унутар CPU-а који користи индексирано адресирање. Нека је инструкција дизајнирана да користи садржај другог регистра (B) као део адресе где се константа налазила. Када CPU прочита opcode, он разуме шта је инструкција и да B садржи део адресе којој се приступа. Такође разуме да се операнд примењује на B да би се формирала стварна адреса којој треба приступити. Када CPU чита операнд, додаје вредност у садржај B, а затим ову нову вредност ставља на адресну магистралу и изводи додатно читање. Прочитана вредност се затим ставља у регистар A CPU-а и инструкција је завршена.

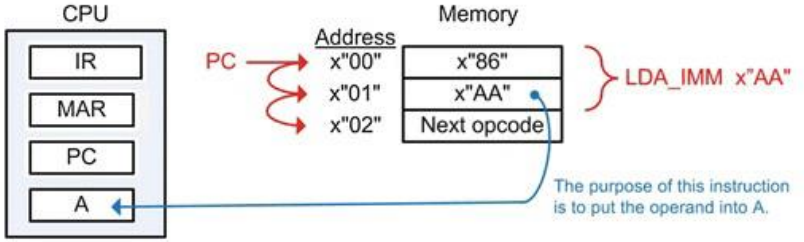
Класе инструкција – Load и Store (LDA_IMM)

Example: Execution of an Instruction to "Load Register A Using Immediate Addressing"

A load instruction using immediate addressing will put the value of the operand into a CPU register. Let's create a program that will load register A in the CPU with the value x"AA". The program is as follows:

Using Mnemonics		Using Hex Values
LDA_IMM x"AA"	or	x"86" x"AA"

When the opcode and operand are put into program memory at x"00", they look like this:



When the CPU begins executing the program, it will perform the following steps:

Step 1 – Fetch the opcode

The program counter begins at x"00", meaning that this address is the location of the first instruction opcode. The PC address is put on the address bus using the MAR and a read is performed. The information read from memory (e.g., the opcode) is placed into the instruction register. The PC is then incremented to point to the next address in program memory. After this step, the IR holds x"86" and the PC holds x"01".

Step 2 – Decode the instruction

The CPU decodes x"86" and understands that it is a "load A with immediate addressing". It also knows from the opcode that the instruction has an operand that exists at the next address location.

Step 3 – Execute the instruction

The CPU now needs to read the operand. It places the PC address (x"01") on the address bus using the MAR and a read is performed. The information read from memory (e.g., the operand) is placed into register A. After this step, A=x"AA". Also in this step, the PC is incremented to point to the next location in memory (x"02"), which holds the opcode of the next instruction to be executed.

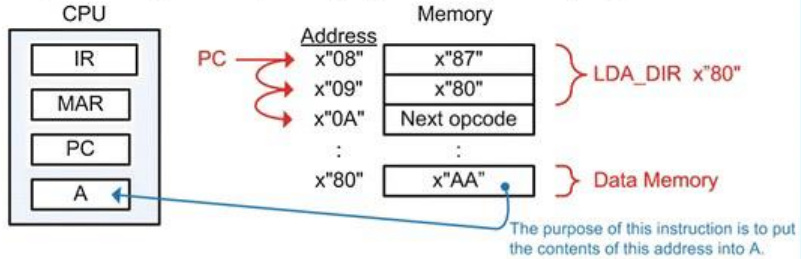
Класе инструкција – Load и Store (LDA_DIR)

Example: Execution of an Instruction to "Load Register A Using Direct Addressing"

A load instruction using direct addressing will put the value located at the address provided by the operand into a CPU register. Let's create a program that will load register A in the CPU with the contents located at address x"80", which has already been initialized to x"AA". The program is as follows:

Using Mnemonics		Using Hex Values
LDA_DIR x"80"	or	x"87" x"80"

When the opcode and operand are put into program memory at x"08", they look like this:



When the CPU begins executing the program, it will perform the following steps:

Step 1 – Fetch the opcode

The program counter begins at x"08", meaning that this address is the location of the instruction opcode. The PC address is put on the address bus using the MAR and a read is performed. The information read from memory (e.g., the opcode) is placed into the instruction register. The PC is then incremented to point to the next address in program memory. After this step, the IR holds x"87" and the PC holds x"09".

Step 2 – Decode the instruction

The CPU decodes x"87" and understands that it is a "load A with direct addressing". It also knows from the opcode that the instruction has an operand that exists at the next address location.

Step 3 – Execute the instruction

The CPU now needs to read the operand. It places the PC address (x"09") on the address bus using the MAR and a read is performed. The information read from memory (e.g., the operand) is the address that contains the value to be put into A. The operand is immediately put on the address bus using the MAR and another read is performed. The value read from address x"80" is placed into register A. After this step, A=x"AA". Also in this step, the PC is incremented to point to the next location in memory (x"0A"), which holds the opcode of the next instruction to be executed.

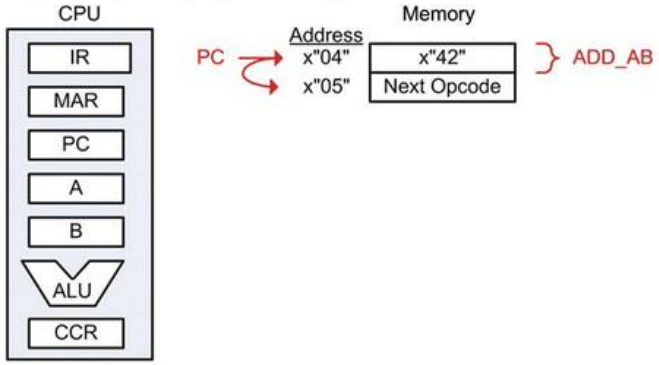
Класе инструкција – манипулација подацима

Example: Execution of an Instruction to "Add Registers A and B"

This instruction adds A and B and puts the sum back into A (A = A+B). This instruction does not require an operand because the inputs and output of the operation reside completely within the CPU. This type of instruction uses inherent addressing, meaning that the location of the information impacted is inherent in the opcode. Let's create a program to perform this addition. The program is as follows:

Using Mnemonics		Using Hex Values
ADD_AB	or	x"42"

When the opcode is put into program memory at x"04", it looks like this:



When the CPU begins executing the program, it will perform the following steps:

Step 1 – Fetch the opcode

The program counter begins at x"04", meaning that this address is the location of the instruction opcode. The PC address is put on the address bus using the MAR and a read is performed. The information read from memory (e.g., the opcode) is placed into the instruction register. The PC is then incremented to point to the next address in program memory. After this step, the PC holds x"05" and the IR holds x"42".

Step 2 – Decode the instruction

The CPU decodes x"42" and understands that it is an "Add A and B". It also knows that there is no operand associated with this instruction.

Step 3 – Execute the instruction

The CPU asserts the necessary control signals to route A and B to the ALU, performs the addition, and places the sum back into A. The CCR is also updated to provide additional status information about the operation.

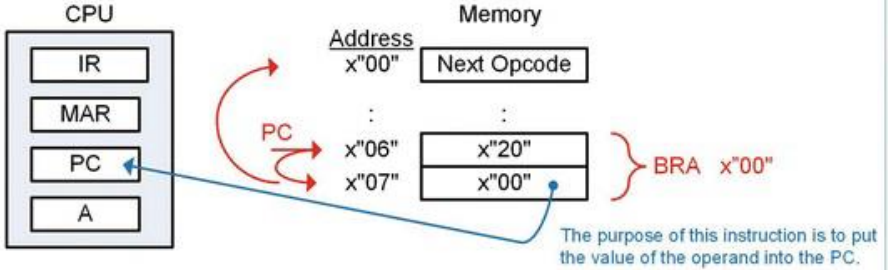
Класе инструкција – гранања

Example: Execution of an Instruction to "Branch Always"

A *branch always* instruction will set the program counter to the value provided by the operand. Let's create a program that will set the program counter to x"00". The program is as follows:

Using Mnemonics		Using Hex Values
BRA x"00"	or	x"20" x"00"

When the opcode and operand are put into program memory at x"06", they look like this:



When the CPU begins executing the program, it will perform the following steps:

Step 1 – Fetch the opcode

The program counter begins at x"06", meaning that this address is the location of the instruction opcode. The PC address is put on the address bus using the MAR and a read is performed. The information read from memory (e.g., the opcode) is placed into the instruction register. The PC is then incremented to point to the next address in program memory. After this step, the PC holds x"07" and the IR holds x"20".

Step 2 – Decode the instruction

The CPU decodes x"20" and understands that it is a "branch always". It also knows from the opcode that the instruction has an operand that exists at the next address location.

Step 3 – Execute the instruction

The CPU now needs to read the operand. It places the PC address (x"07") on the address bus using the MAR and a read is performed. The information read from memory (e.g., the operand) is the address to load into the PC. The operand is latched into the PC and the instruction is complete. After this instruction, the PC=x"00" and the program will begin executing instructions at that address.