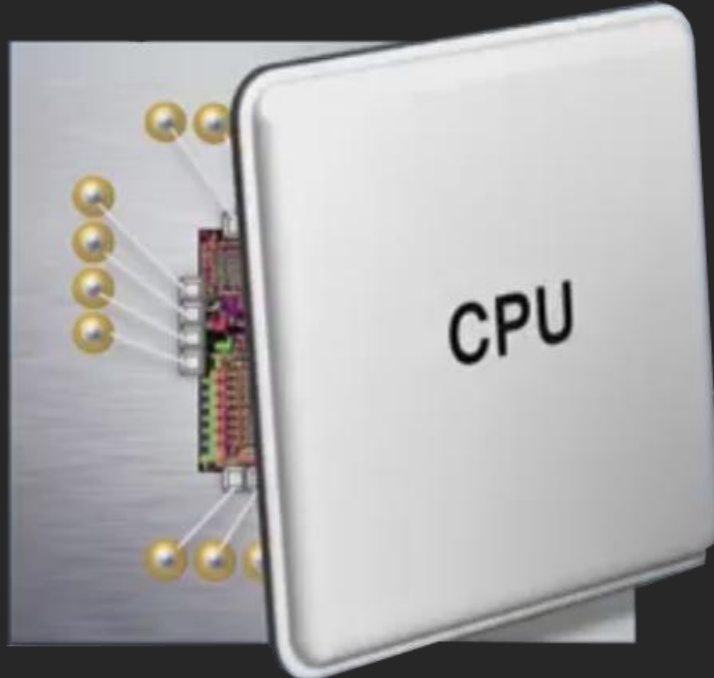


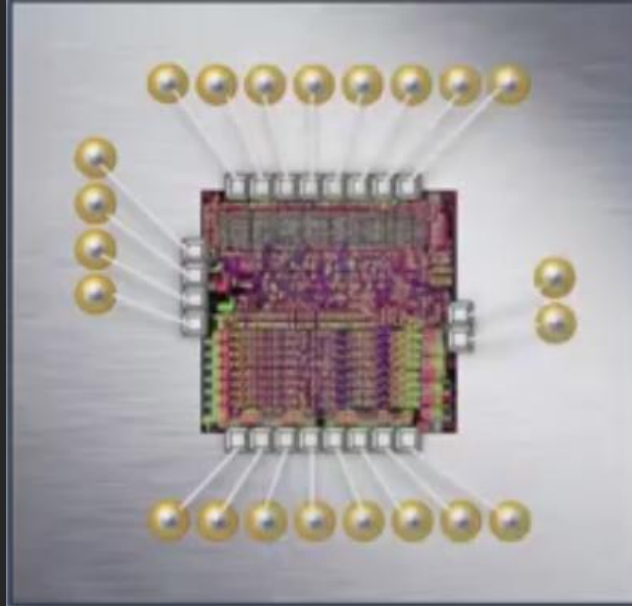
## CPU တွေဘယ်လို အလုပ်လုပ်သလဲ



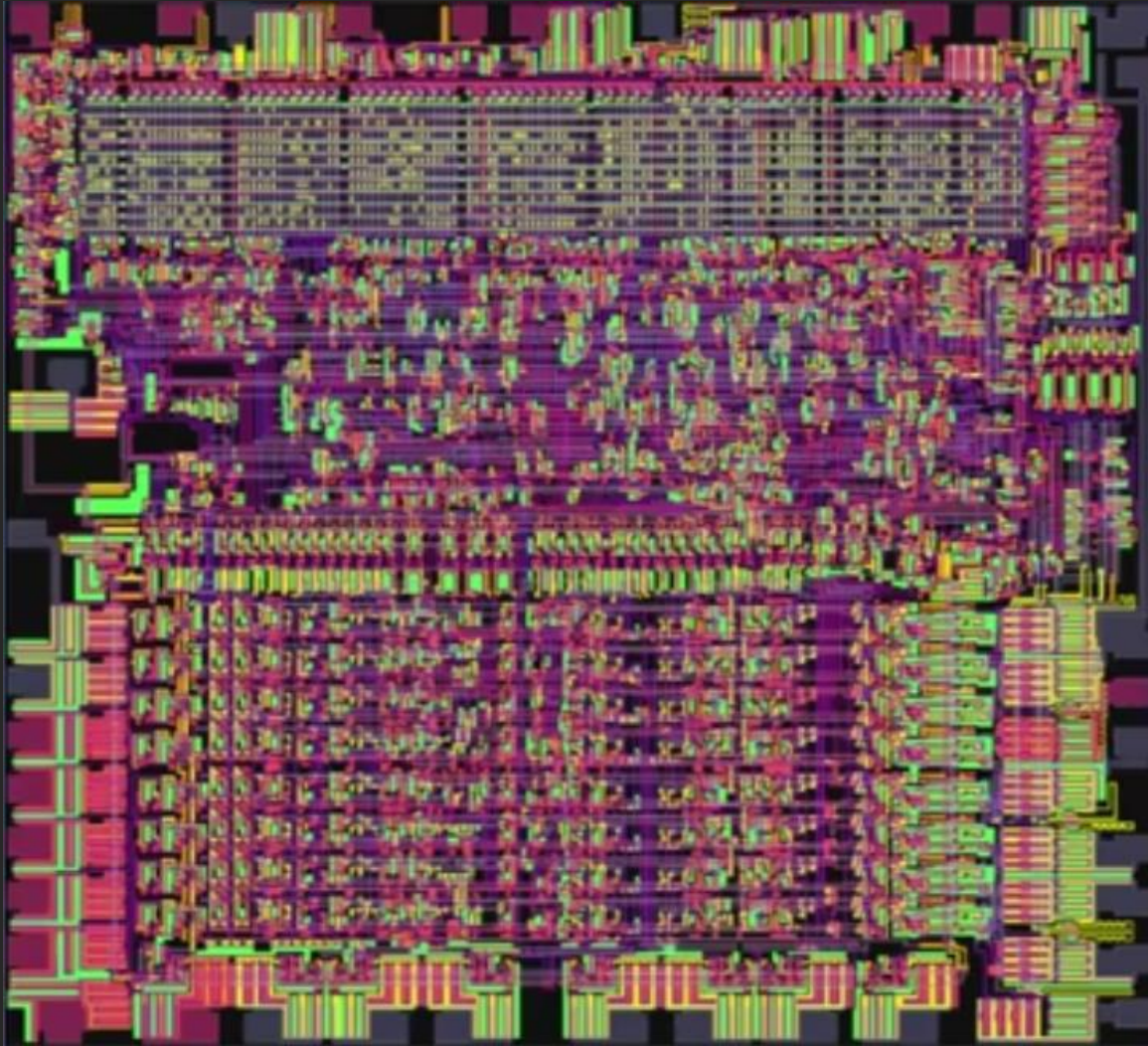
CPU ရဲ့ မူရင်း စာလုံးက တော့ Central Processing Unit ဖြစ်ပါတယ်။ သူ့ကို ကွန်ပျူတာ ဦးနှောက်လို့လဲ အမိပါယ် ဖွင့်ဆိုကြပါတယ်။ CPU ဘယ်လို အလုပ်လုပ်သလဲ ဆိုတာကို သိရင် ကွန်ပျူတာ ဘယ်လို အလုပ်လုပ် သလဲ ဆိုတာကို လဲ ကောင်းကောင်း သဘောပေါက်သွားမှာပါ။

အခု CPU ကို ရဲ့ အဖုန်းကို ဖွင့်ကြည့်ရအောင်





အထက်က ပုံကတော့ CPU ရဲ့ အဖုန်းကို ခွဲလိုက်ရင် အတွင်းပိုင်းကို မြင်ရမယ့်ပုံပါ။ အခု ပုံကြီးချဲ့လိုက်မယ်၊  
နည်းနည်းပိုမြင်နိုင်အောင်လို့



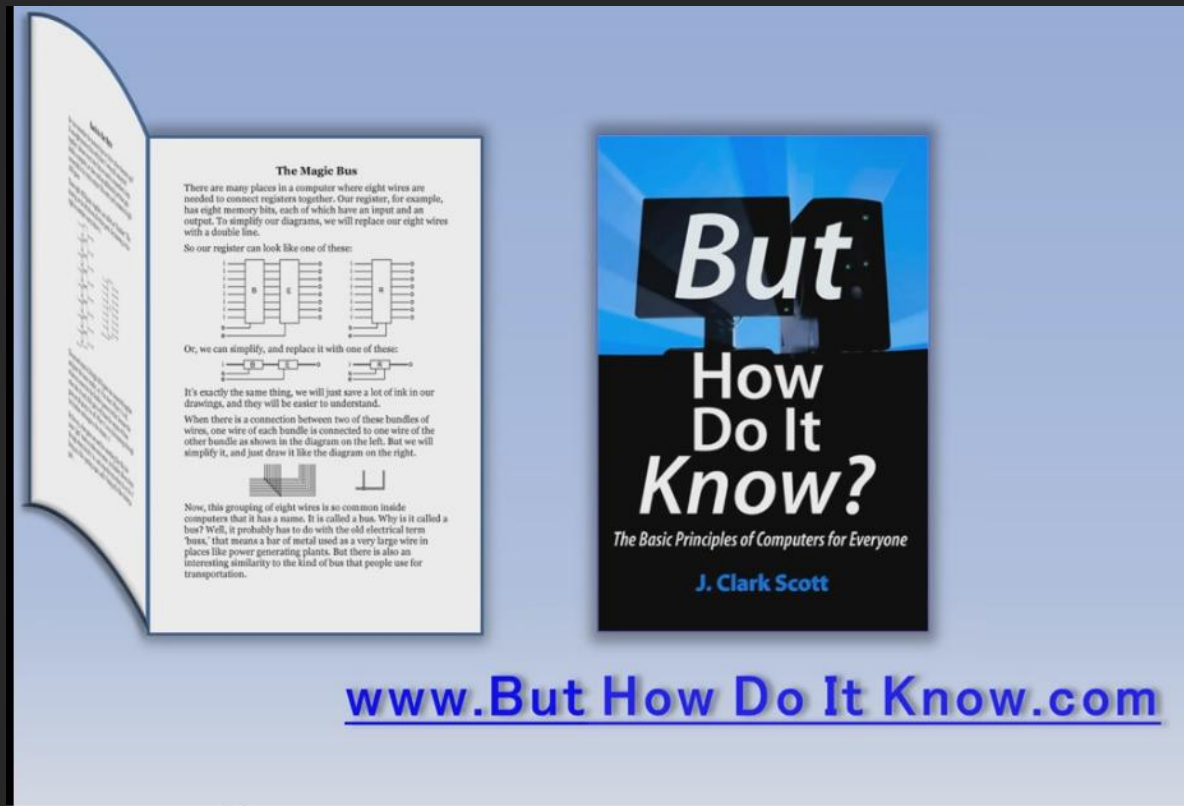
CPU အထဲမှာ ဂါယာ အမျိုးမျိုး က သတင်းအချက်အလက်တွေ သယ်ယူပို့ဆောင်တဲ့ အလုပ်ကိုလုပ်နေပါတယ်၊ အခု Post မှာ ရှင်းပြဖို့ အသုံးပြုသွားမယ့် CPU ကတော့ 65-02 လို့ခေါ်ပါတယ်၊ ဒီ CPU အမျိုးအစားကို Apple အပါအဝင် ကွန်ပျူတာတွေအများစု မှာ အသုံးပြုကြပါတယ်၊ ဒီ CPU ကိုရဲ့ အသေးစိတ် ဖြုတ်တပ်နည်းကိုသိချင်ရင်တော့ [Virusual6502.org](http://Virusual6502.org) ဝက်ဆိုက်မှာ သွားရောက်ကြည့်ရှုနိုင်ပါတယ်၊ ဘယ် CPU မှာ မဆို CPU အတွင်း လုပ်နေတဲ့အလုပ်တွေကို အခြေအနေမှန်မှန်နဲ့ တည်ငြိမ်စွာ Data သယ်ဆောင်မှု လုပ်ဖို့ အတွက် တိကျတဲ့ အချိန် အတိုင်း အတာ တစ်ခုတိုင်းမှာ ဖွင့်လိုက် ပိတ်လိုက် အလုပ်လုပ်နေတဲ့ ဂါယာတစ်ချောင်းရှိပါတယ်၊ အဲ့ဒီ ဂါယာကပဲ CPU အတွင်း လုပ်ဆောင်မှုတွေကို မျှတစေပါတယ်၊ အဲ့ဒီ ဂါယာကို Clock လို့ခေါ်ပါတယ်၊ Disimulation မှာ တစ်စက္ကန့်ကို Clock က နှစ်ကြိမ်ဖွင့် ပီး နှစ်ကြိမ်ပိတ်ပါတယ်၊ နောက်ပိုင်းပေါ်တဲ့ CPU တွေမှာတော့ Clock ရဲ့ အလုပ်လုပ်နှုန်းကို GHz (GigaHertz) နဲ့ တိုင်းတာပါတယ်၊ Giga ကတော့ 1 billion ကို ညွှန်းပီးတော့ Hz Hertz ကတော့ တစ်စက္ကန့် အတွင်း ဘယ်နှစ်ကြိမ်အလုပ်လုပ်လဲကိုညွှန်းပါတယ်၊ ဒီတော့ အခုခါတ် CPU တွေက တစက္ကန့်ကို အကြိပ် ရေး Billion

အတော်များများ ဖွင့် လိုက် ပိတ်လိုက်ဖြစ်နေပါတယ်။ ဒီလို တစ်စက္ကန့်မှာ Billions အတော်များများ ဖွင့် ပိတ်နိုင်ခြင်းကပဲ CPU ရဲ့ တွက်ချက် အလုပ်လုပ်နိုင်မှု စွမ်းအားဖြစ်ပြီး CPU ကို အလွန် ရှုတ်ထွေးတဲ့ တွက်ချက်မှုတွေကို လုပ်ကိုင်နိုင်စေပါတယ်။

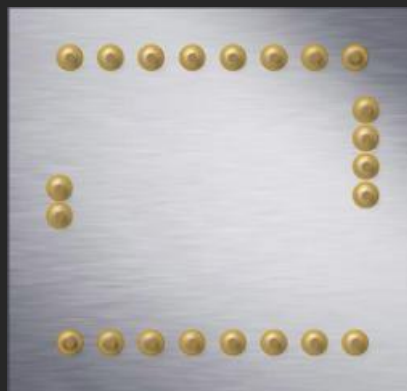
ဒါပေမယ့် Clock ပွင့် တဲ့ အချိန်မှာ CPU ရဲ့ အလုပ်လုပ်ပုံကတော့ လူတွေထင်ထားသလို ရှုပ်ထွေးတာ မဟုတ်ပဲ ရိုးရိုး ရှင်းရှင်းလေးပဲ အလုပ်လုပ်သွားတာပါ။ အဲ့ဒီ အလုပ်လုပ်ပုံကိုပဲ ဒီ Post မှာလေ့လာသွားမှာပါ။

CPU ကို အထူးထုတ်လုပ်တဲ့ ကုမ္ပဏီ နှစ်ခုကတော့ Intel နဲ့ AMD တို့ဖြစ်ပါတယ်။ ဒီမှာ အသေးစိတ်လေ့လာသွားမယ့် CPU ရဲ့နာမည်ကိုတော့ Scott CPU လို့ခေါ်ပါတယ်။ တကယ်တော့ Scott CPU ဆိုတာ အပြင်မှာတကယ် မရှိပါဘူး။ လေ့လာသူတွေအတွက် အဆင်ပြေအောင် Clerk Scott က သူ့ရဲ့ “But How to it Know” ဆိုတဲ့စာအုပ်မှာ တည်ဆောက်ပြထားတဲ့ စာအုပ်ထဲက လေ့လာရေး CPU သာဖြစ်ပါတယ်။ Scott CPU ရဲ့ ဒီဇိုင်းက Clerk Scott ရဲ့ Copyright ဖြစ်ပြီး Youtube Master က ရှင်းပြလို သူရှင်းပြတဲ့ အတိုင်း ဒီ Post မှာ ပြန်ဖော်တာပါ။ Hotdoitknow.com ဝက်ဆိုက်မှာ ဒီစာအုပ်ကိုဝယ်ဖက်နိုင်ပါတယ်။ ဒီစာအုပ်ကတော့ တကယ့်ကို ဂျွတ်လှပါတယ်။ အသေးစိတ်ကို တဆင့်စီ ဖြေးဖြေး ရှင်းသွားမိး

ဖတ်ရှုသူကို ခက်ခဲစေမယ့် ဘယ်လို Technical Jarcon (နည်းပညာသုံးသီးသန့်စာလုံး) တွေမပါပါဘူး။



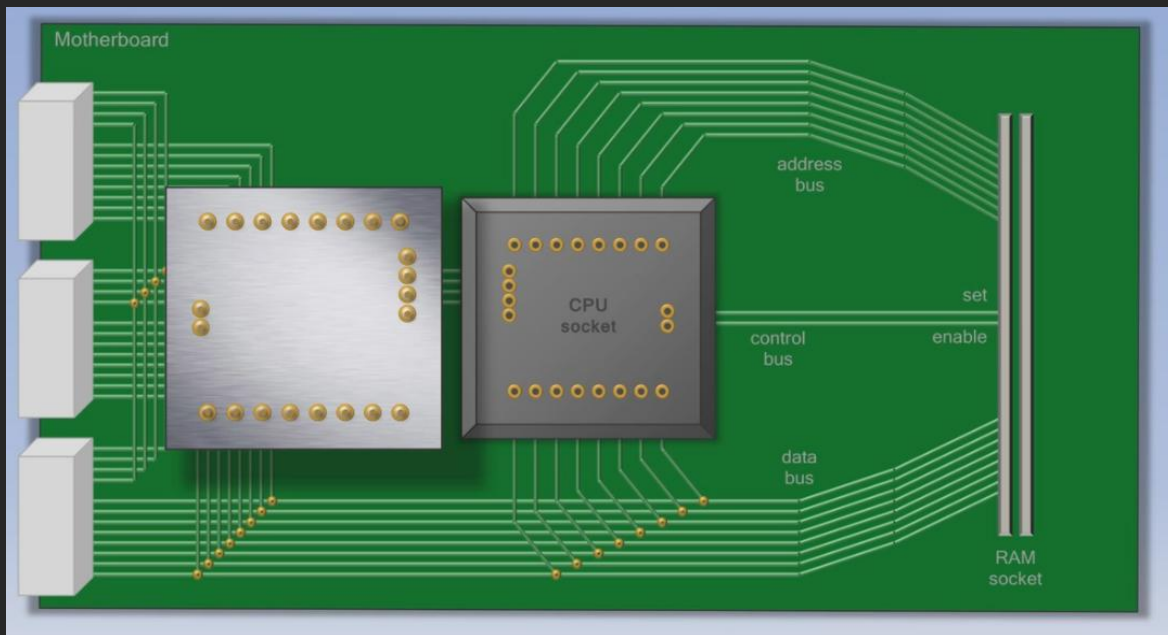
အခု CPU ကိုပြောင်းပြန်လှန်ပီးအောက်ပိုင်းကို တစ်ချက်ကြည့်ကြမယ်။



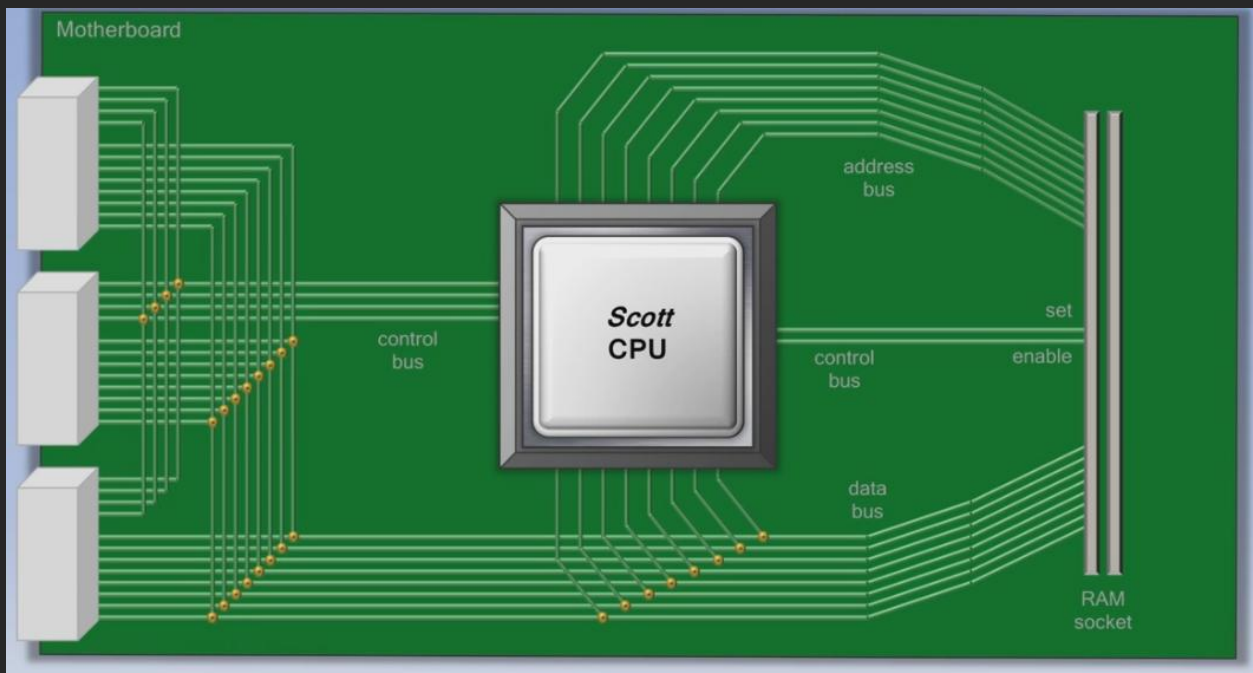
Pins တွေအတော်များများ အပြင်ထွက်နေတာကို တွေ့ရပါမယ် ၊ အဲ့ဒီ Pin တွေကပဲ သတင်းအချက်အလက်တွေကို CPU ထဲကို ပို့ပီး အဲ့ဒီ Pin တွေကပဲ အချက်အလက်တွေကို CPU ကနေ အပြင်ကို ပြန်ထုတ်ပေးပါတယ်။ ဒီ Pin တွေက သတင်းအချက်အလက်တွေအတွက် CPU ထဲကို ဝင်မယ့်



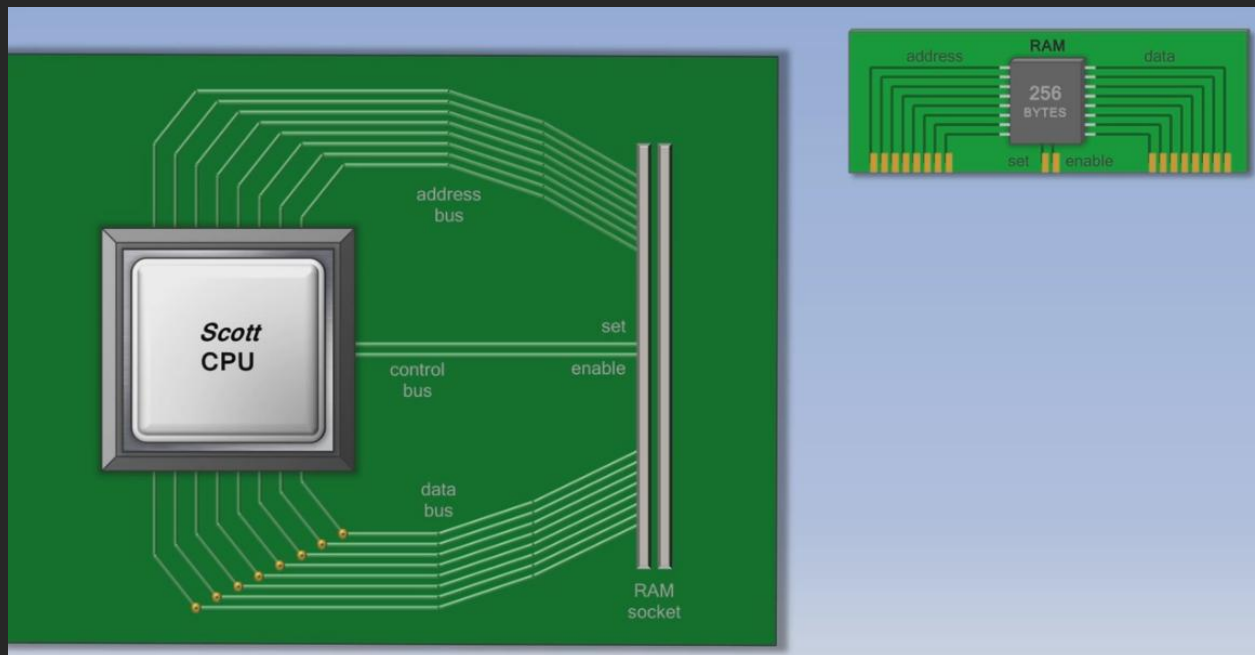
၎င်းပေါက် ထွက်ပေါက်တွေပေါ့၊ CPU ကို Mother Board ထဲမှာ စိုက်ထည့်ထားပါတယ်၊ အောက်က အတိုင်းပေါ့၊



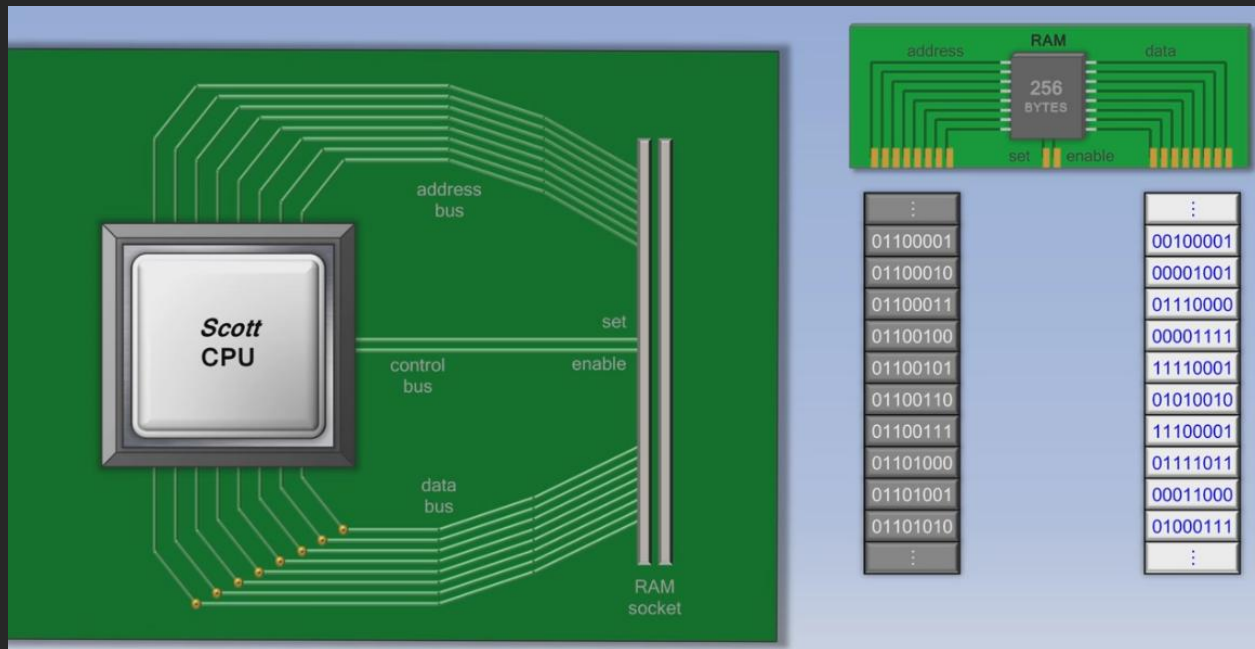
MotherBoard ကတော့ ကွန်ပျူတာရဲ့ Components တွေအားလုံးကို တပ်ဆင်ပီး အားလုံး အချိတ်အဆက်မိစေတဲ့ Circuit Board တစ်ခုဖြစ်ပါတယ်၊ အထက်က ပုံက CPU အဖုံးကို ဖွင့်ထားတဲ့ ပုံပါ အခု ပြန်ပိတ်လိုက်ကြပီး MotherBoard ထဲပြန်ထည့်ကြမယ်၊ အောက်က အတိုင်း



အထက်ကပုံရဲ့ ညာဘက်မှာတော့ RAM ထည့်မယ့် Socket ပေါက်ကိုတွေ့ရပါမယ်။ RAM ကတော့ Random Access Memory ရဲ့ အတိုကောက်ပါ။ CPU ကနေ တွက်ချက် အလုပ်လုပ်လို့ရတဲ့ Data တွေအားလုံးကို RAM ကိုလှမ်းပို့ပီး RAM ထဲမှာ သိုလှောင်ထားပါတယ်။ ဒီနေရာမှာ RAM နဲ့ CPU တို့ ဘယ်လို တွဲဖက်ပီး အလုပ်လုပ်လဲ သိဖို့လိုအပ်တဲ့ အတွက် RAM ဘယ်လို အလုပ်လုပ်လဲဆိုတာကိုလဲ လေ့လာရမယ်။ အခု အထက်က ပုံရဲ့ ဘယ်ဘက်က မလိုအပ်တဲ့ Wire အစိတ်အပိုင်းတွေကို ဖြတ်ထုတ်ပီး RAM Chips ပါပါဝင်မယ့် ပုံကို ကြည့်ကြမယ်။



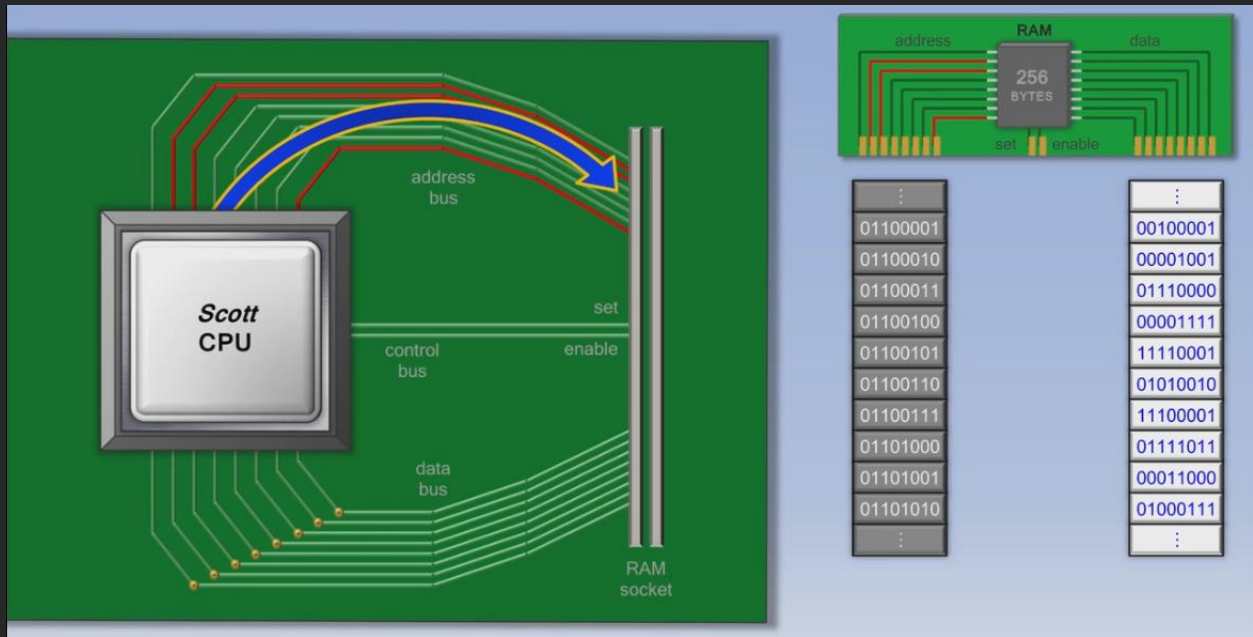
RAM ထဲမှာ Address လို့ခေါ်တဲ့ လိပ်စာတွေအများကြီးထည့်ထားတဲ့ စာရင်းပါပါတယ်။ Address တစ်ခုစီက Data အစိတ်အပိုင်းတစ်ခုကိုညွှန်းပါတယ်။ အောက်က ပုံမှာ Addresses ကော Data တွေကိုပါ သရုပ်ဖော်ပြထားတယ်။



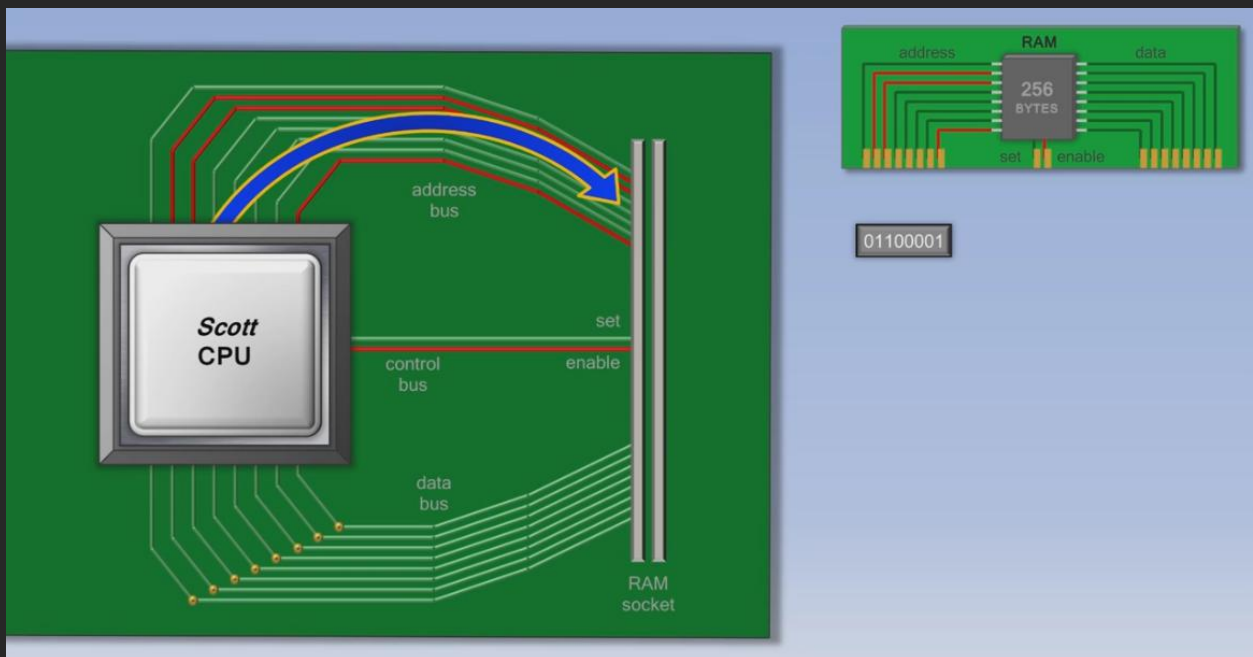
CPU က RAM ထဲက Data တစ်ခုစီကို Request (တောင်းယူ) ပီးတော့ အလုပ်လုပ်ပါတယ်။ Data တစ်ခု တောင်းယူ အလုပ်လုပ်ပီး နောက်တစ်ခု လုပ်တယ် နောက်တစ်ခုပီး နောက်တစ်ခု အလုပ်လုပ်တယ် စသဖြင့် တစ်ခုပီးနောက်တစ်ခု အစီအစဉ်အလိုက် အလုပ်လုပ်သွားတယ်။ အစီအစဉ်တစ်ကျ အလုပ်လုပ် ဆိုပေမယ့် အမြဲ တမ်းတော့ အစီအစဉ်တစ်ကျ အလုပ်လုပ်နေမှာ မဟုတ်ပါဘူး။ CPU ကို အစီအစဉ်တစ်ကျမဟုတ်ပဲ RAM ကနေ Data တွေကို တောင်းယူ အလုပ်လုပ်အောင် ညွှန်ကြားနိုင်ပါတယ်။ RAM က လဲ ဒီလို အစီအစဉ်မကျတဲ့ Request (တောင်းဆိုမှု) မျိုးကို လဲ လက်ခံနိုင်ပါတယ်။ ဒါ့ကြောင့်ပဲ သူ့ကို Random Access Memory လို့ခေါ်တာပါ။ ဒီတော့ CPU အနေနဲ့ ပုံမှန်အတိုင်းဆိုရင် RAM ထဲက ဒေတာတွေကို အစီအစဉ်အလိုက် တောင်းယူ အလုပ်လုပ်တယ်ဆိုပေမယ့် လိုအပ်လာရင် အစီအစဉ်မကျတဲ့ ပုံစံနဲ့ စိတ်ကြိုက် Data ကို တောင်းယူ အလုပ်လုပ်နိုင်တယ်ဆိုတာကိုလဲ သဘောပေါက်သင့်ပါတယ်။

Computer က Program တစ်ခုကိုစ Run ပီဆိုတာနဲ့ CPU က RAM ဆီကနေ အလုပ်လုပ်လိုတဲ့ Program ကို တောင်းတဲ့ Request ကို တောင်းဆိုလိုက်ပါတယ်။





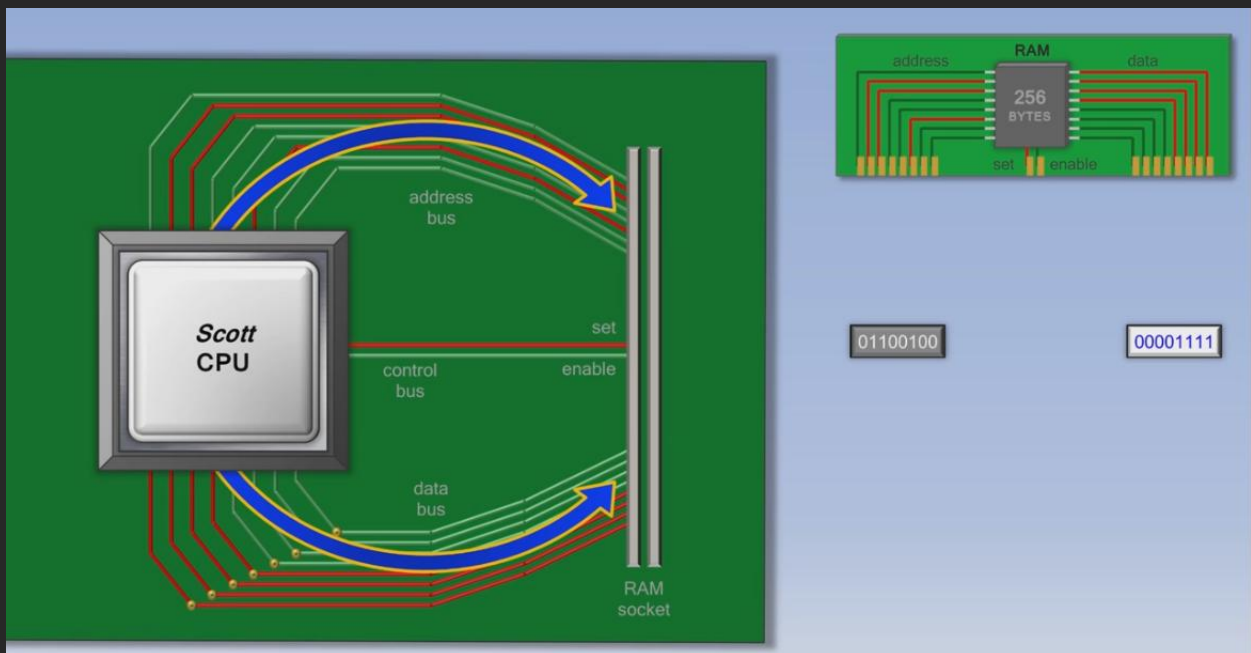
RAM address မှာ တော့ 0 နဲ့ 1 တွေသာပါဝင်တဲ့ ဂဏန်းတွေတန်းစီထားပါတယ်။ အဲ့ဒီဂဏန်းတစ်ခုစီက တွေ On မလား Off မလား ဆိုတဲ့ အမိန့်ကို ကိုစားပြပါတယ်။ RAM မှာ Address တွေမရှိရင် RAM ကလဲ ဘာမှလုပ်နိုင်တော့ မှာမဟုတ်ပါဘူး။ RAM အလုပ်လုပ်ဖို့ဆို CPU ကလဲ Enable Wire ကို On ပေးရပါတယ်။ အောက်က ပုံမှာ Enable Wire On နေတာကို ကြည့်ပါ။ (မျက်စိရှင်ရှင်ထား)



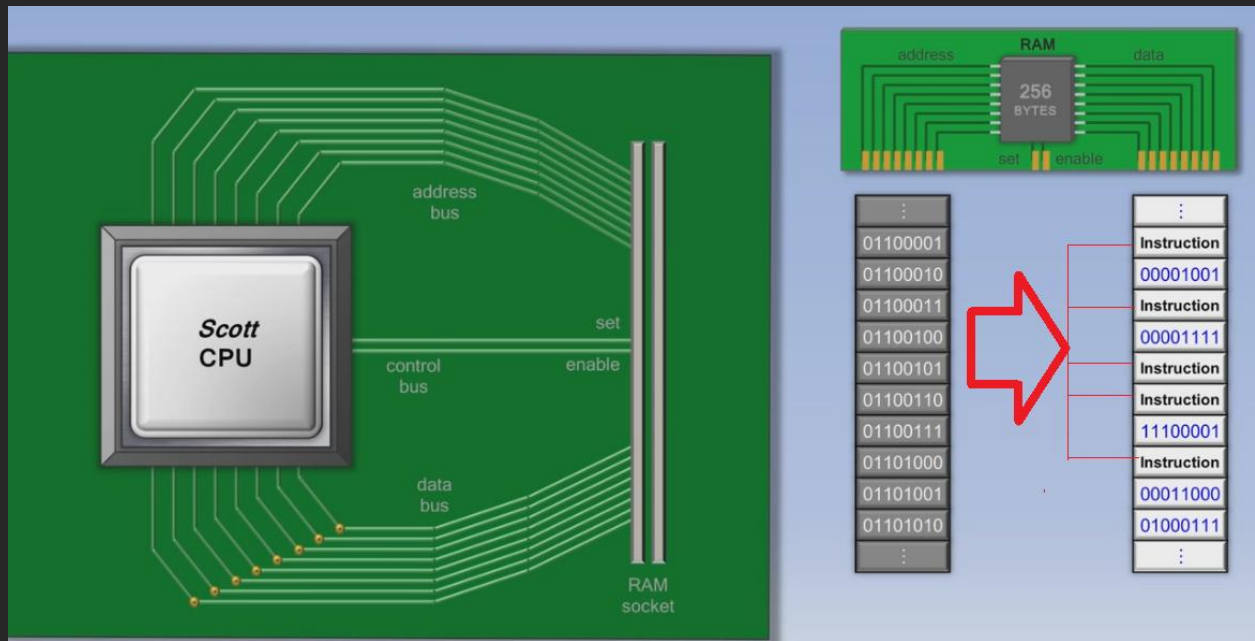
အကယ်၍ Address လဲရှိမယ် Enable Wire ကလဲ On နေမယ်ဆိုရင်တော့ RAM က CPU ကတောင်း ဆိုထားတဲ့ Address မှာရှိတဲ့ Data ကို CPU ကိုပြန်ပို့ပေးပါတယ်။ ဒီလို Data ပြန်ရပီဆိုရင်တော့ CPU က အဲ့ဒီ

ဒေတာကို ကို အလုပ်စလုပ်တော့တယ်။ အလုပ်လုပ်လို့ပီးသွားရင် CPU က RAM ဆီကို နောက်ထက် Address တစ်ခု ပြန်ပို့မယ်။ Enable Wire ကို On လိုက်မယ်။ ဒါဆိုရင် RAM က နောက်ထက် Data တစ်ခုကို CPU ကို ဆက်ပို့ပေးမယ်။ CPU က လုပ်ဆောင်ချက်တွေလုပ်မယ်။ ပီးရင် Address အသစ်ကို RAM ကို ပြန်ပေးမယ်။ စသဖြင့် အလုပ် Data ပို့လိုက် ယူလိုက်နဲ့ ဆက်တိုက် အလုပ်လုပ်သွားမယ်။ ဒီဖြစ်စဉ်က ကွန်ပျူတာမှာ အမြဲ ဖြစ်နေ တဲ့ ဖြစ်စဉ်ဖြစ်ပါတယ်။

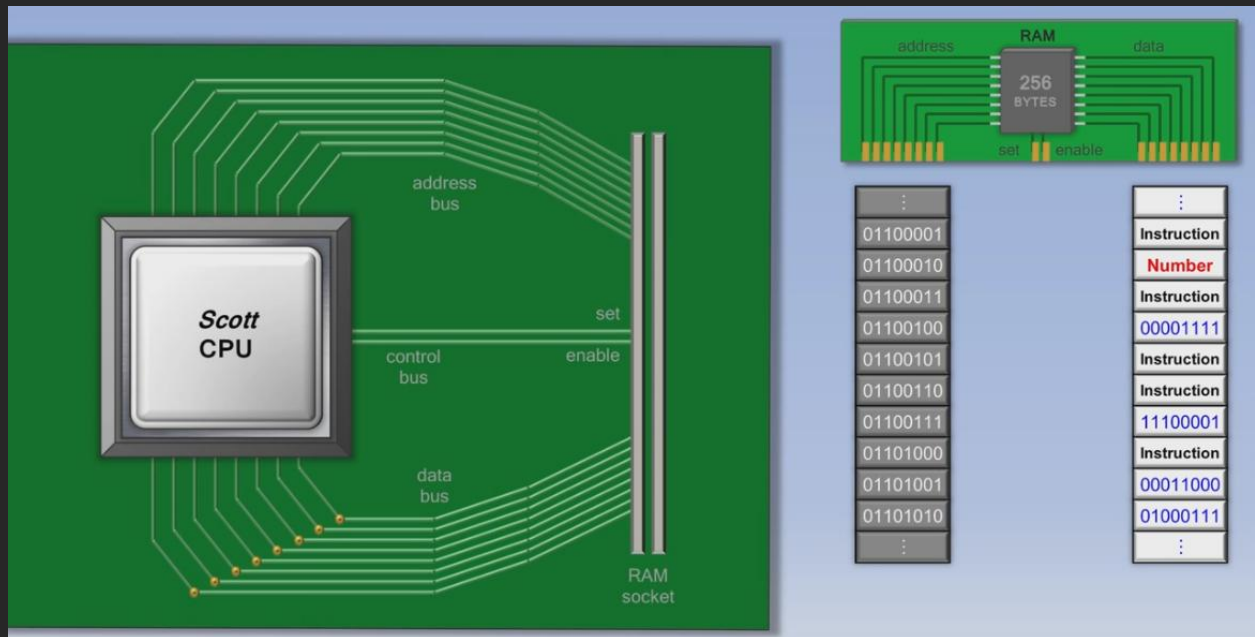
အထက်က ဖြစ်စဉ်က CPU Data Processing လုပ်တဲ့ဖြစ်စဉ်ပါ။ တကယ်၍ CPU က Data တစ်ခုကို အလုပ်လုပ်ပီးသွားလို့ အဲ့ဒီ အလုပ်နဲ့ ပက်သက်တဲ့ Data ကို သိမ်းဖို့ပါလို့လာပီဆိုရင်တော့ အထက်က နည်း အတိုင်း Address သာလျှင် RAM ကို ပြန်ပေးတာမဟုတ်တော့ပဲနဲ့ Address ကော Data ကော ကိုပြန်ပေးတယ်။ Address ကို Address Bus ကနေ ပြန်ပေးတယ်။ Data ကို Data Bus ကနေ ပြန်ပေးတယ်။ ဒီလို Address ကော Data ကော ပြန်ပေးတဲ့အခါ Set Wire ကိုလဲ On ထားပါတယ်။ ဒါဆိုရင်တော့ CPU ကပြန်ပေးလိုက်တဲ့ Data ကို RAM က ပြန်ပေးလိုက်တဲ့ Address အတိုင်း ရှိရင်းစွဲ Data ကို Override လုပ်ပစ်ပီးသိမ်းလိုက်ပါတယ်။



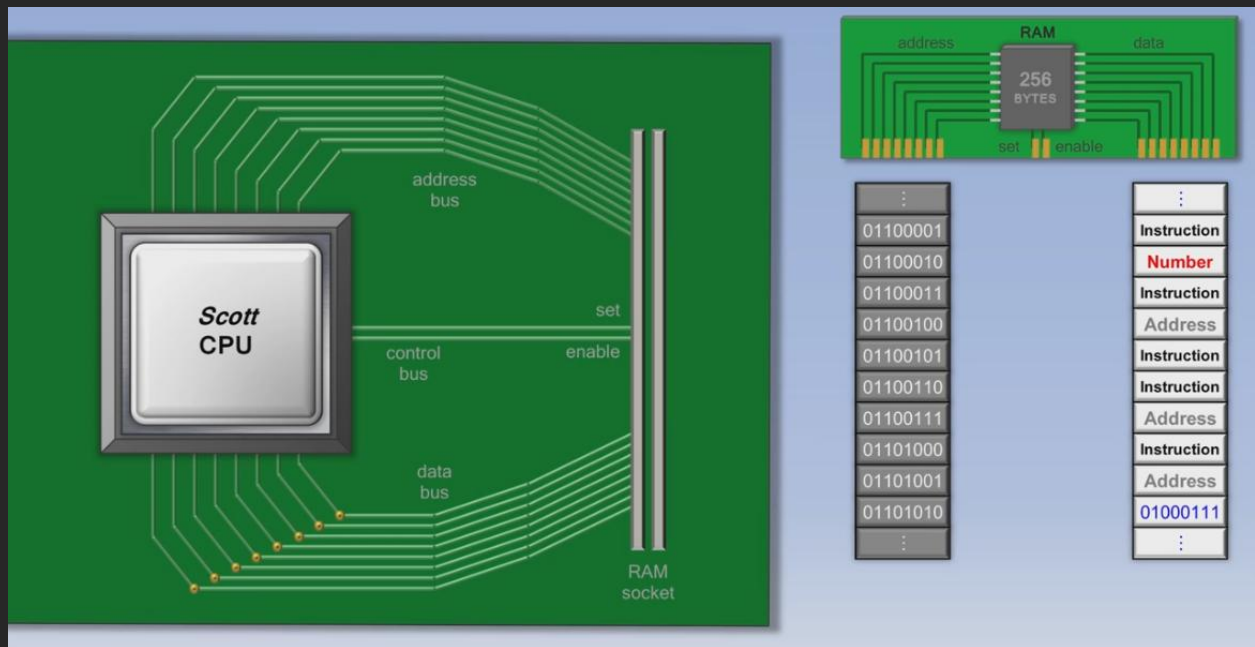
RAM ထဲမှာ ရှိတဲ့ 0 တွေ 1 တွေက တကယ်တော့ မတူညီတဲ့ အချက်အလက်တွေကို ကိုယ်စားပြုပါတယ်။ အရေး အကြီးဆုံး အချက်ကတော့ Instructions (ညွှန်ကြားချက်တွေ) ပါ။ Instructions တွေက CPU ကို ဘာအလုပ်ကို လုပ်မလဲ ဆိုတာကိုညွှန်ကြားပေးတယ်။ အောက် အနီရောင်မြားပြထားတဲ့ အနီရောင်မျဉ်းတန်း တွေအားလုံး က ညွှန်ကြားချက် (Instruction) တွေပါ။



Data တွေထဲမှာ Numbers တွေလဲပါပါတယ်။ ဒီ Numbers တွေက Compare(နိုင်ယှဉ်ဖို့) Add (ပေါင်းဖို့) စသဖြင့် လုပ်ချင်တဲ့အလုပ်ကိုလုပ်နိုင်ဖို့ပါ။

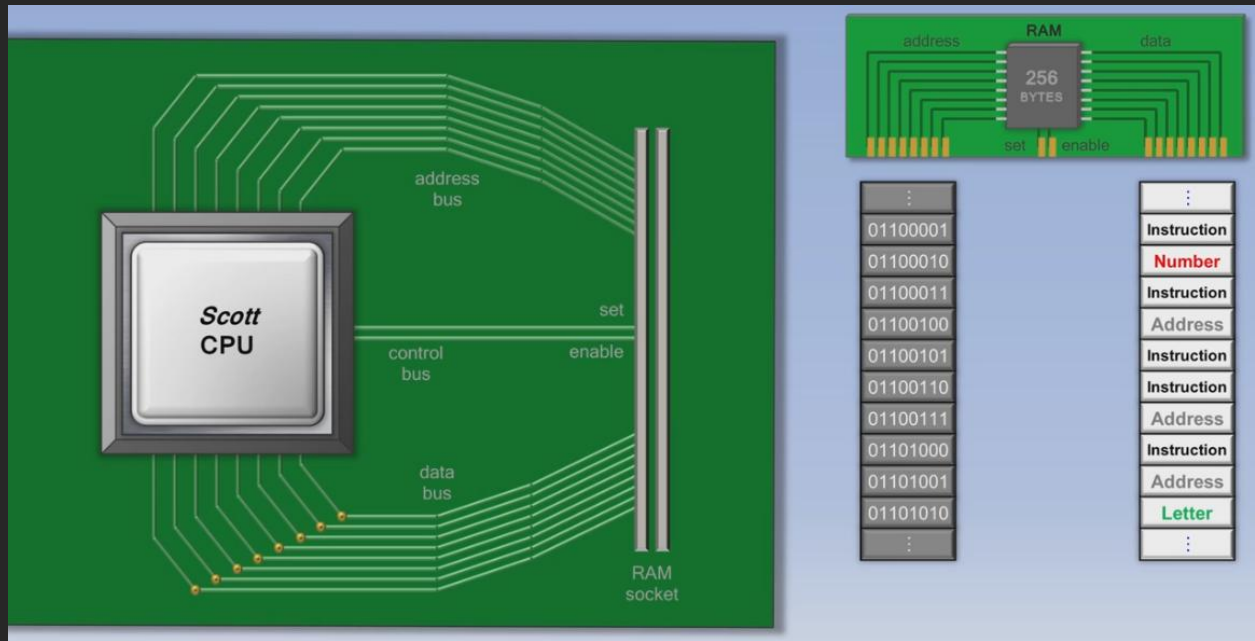


RAM ထဲမှာ Addresses တွေလဲပါပါသေးတယ်။ Address တွေက ညွှန်းတဲ့ Data တွေထဲမှာ Address တွေကို ပြန်သိမ်းထားတယ်ဆိုတော့ နည်းနည်းတော့ ကြောင်ချင်စရာ ဖြစ်သွားမယ်။ ဒါပေမယ့် ဒီ Address တွေကလဲ အသုံးဝင်လှသလို လိုအပ်တဲ့ အချက်တွေမျိုးစုံကို သိုလှောင်ထားနိုင်ပါတယ်။



ဥပမာ OutPut Device တစ်ခုစီကို Data ကို ထုတ်ပေးချင်တယ်ဆိုပါစို့ ၊ ဒါဆိုရင် ကွန်ပြူတာထဲမှာရှိတဲ့ အဲ့ဒီ Device ရဲ့ Address ကို သိဖို့လိုပါတယ်၊ ကွန်ပြူတာနဲ့ ချိတ်ဆက်ထားတဲ့ဖုန်းကို Data Output လုပ်ပေးချင်ရင် ကွန်ပြူတာနဲ့ ချိတ်ဆက်တာနဲ့ ဖုန်းကို Register လုပ်ထားတဲ့ Address ကိုသိဖို့လိုပါတယ်၊

RAM ထဲမှာ Latter တွေကိုလဲ သိလျှောင်ပါသေးတယ်၊ အကယ်၍ Monitor Screen ပေါ်ကို စာသားတွေ Output လုပ်ချင်တယ်ဆိုရင် အဲ့ဒီ Output လုပ်လိုတဲ့ Data တွေကို 1 နဲ့ 0 တွေသုံးပြီး RAM ထဲမှာသိလျှောင်ထားပါတယ်၊ စာသားတစ်လုံးစီကို သက်ဆိုင်ရာ Character Code အလိုက် 1 နဲ့ 0 တွေပေါင်းထားတဲ့ သီးသန့်အုပ်စု အနေနဲ့ သိလျှောင်တယ်၊ Character Code တွေကတော့ Obitary တွေဖြစ်ပါတယ်၊ ဥပမာ 01100001 က a အက္ခရာ အသေး 01000111 က G အက္ခရာ ဆိုပီး သတ်မှတ်တာပါ။



အထက်က ဖော်ပြခဲ့တဲ့ Instruction, Number, Address, Letter တွေကတော့ RAM ထဲ က Data အနေနဲ့ ထည့်သွင်းသိမ်းဆည်းတဲ့ Data တွေဖြစ်ပါတယ်။

အခု CPU ရဲ့ Instruction Set ကို တစ်ချက်ကြည့်ကြမယ်။





## Instruction Set

**LOAD** a number from RAM into the CPU

**ADD** two numbers together

**STORE** a number from the CPU back out to RAM

**COMPARE** one number with another

**JUMP IF *Condition*** to another address in RAM

**JUMP** to another address in RAM

**OUTPUT** to a device such as a monitor

**INPUT** from a device such as a keyboard

RAM ရဲ့ Data တွေထဲမှာ Instruction တွေပါတာကို သိနေလောက်ပါပြီ။ အဲ့ဒီ Instruction တွေ ဘယ်လောက် အထိ အရေးကြီးလဲ ဆိုတာလဲ ပြောခဲ့ဖူးပါဘူး။ အထက်က ပုံကတော့ အသုံးများတဲ့ Instructions တွေကို ဖော်ပြထားတဲ့ပုံပါ။ ဒါအကုန်တော့မဟုတ်သေးဘူး။ အသုံးအများဆုံးလို့ပဲပြောတာပါ။ အခု အဲ့ဒီ Instruction တွေကို တစ်ခုစီရှင်းမယ်။

LOAD : : Load Instruction ကတော့ RAM ကနေ ဂဏန်းတန်ဖိုး တစ်ခုကို CPU ထဲကို ဝန်တင်တဲ့ အခါ မှာ သုံးတဲ့ Instruction ပါ။

ADD : : Add Instruction ကတော့ ဂဏန်း တန်ဖိုး နှစ်ခုကို ပေါင်းဖို့ အတွက် သုံးတဲ့ Instruction ပါ။

STORE: : Store Instruction ကတော့ Add Instruction ညွှန်ကြားချက်အတိုင်း ပေါင်းလို့ရလာတဲ့ Data ကို CPU က နေပြန်ပို့တဲ့အခါမှာ RAM မှာ ပြန်သိမ်းဖို့ အတွက်သုံးပါတယ်။

COMPARE : : Compare Instruction ကတော့ ဂဏန်းတန်ဖိုး နှစ်ခုမှာ ဘယ်တစ်ခုပိုကြီးလဲနဲ့ သူတို့ တန်ဖိုး တူမတူ စစ်လိုတဲ့အခါသုံးပါတယ်။ Compare Instruction က JUMP IF Instruction နဲ့ အများစု တွဲသုံးပါတယ်။

JUMP IF : : JUMP IF Instruction ကိုတော့ အကယ်၍ စစ်ဆေးတဲ့ အခြေအနေတစ်ခုက မှန်ရင် Address A ကိုသွားမယ် မှား ရင် ဘာမှမလုပ်ဘူး စသဖြင့် အခြေအနေကိုစစ်ဆေးပြီး Address ကို Randomly ပြောင်းချင်တဲ့အခါမှာ သုံးပါတယ်။

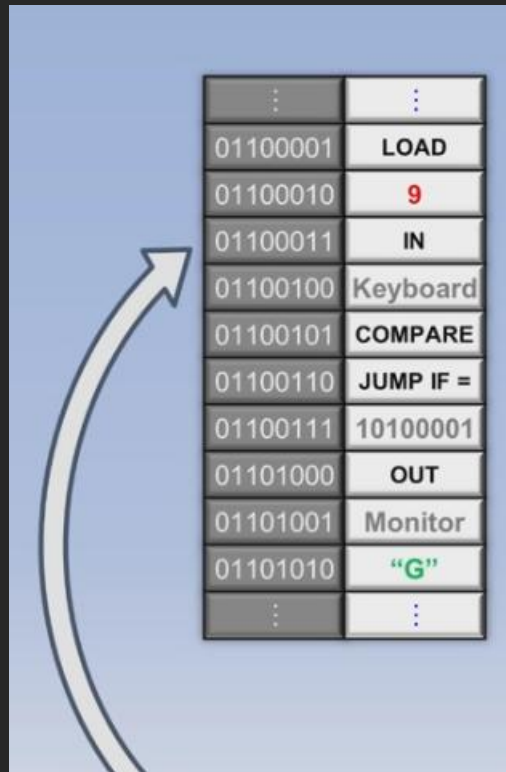
JUMP : : Jump ကတော့ RAM ထဲက Address တွေကို အစီအစဉ် မကျတဲ့ ပုံနဲ့ စိတ်ကြိုက် ရွေးချယ်လို တဲ့အခါမှာသုံးတယ်။

OUT : : Out Instruction ကတော့ Data တွေကို Monitor သို့ အခြား External Devices တွေကို Output ထုတ်လိုတဲ့အခါသုံးပါတယ်။

IN : : IN Instruction ကိုတော့ Keybaord လိုမျိုး External Device ကနေ Data ထည့်သွင်းတာကို လက်ခံ တဲ့ အခြေအနေမျိုးမှာသုံးပါတယ်။

အထက်က အချက်တွေကို သင်သေပြဖို့ အတွက် Gussing Program

လေးတစ်ခုကို တည်ဆောက်ကြည့်ပါမယ်။ အောက်က ပုံကိုသေချာကြည့်ပါ မျက်စိကို ရှင်ရှင်ထားဖို့လိုမယ်။



⋮	⋮
01100001	LOAD
01100010	9
01100011	IN
01100100	Keyboard
01100101	COMPARE
01100110	JUMP IF =
01100111	10100001
01101000	OUT
01101001	Monitor
01101010	"G"
⋮	⋮

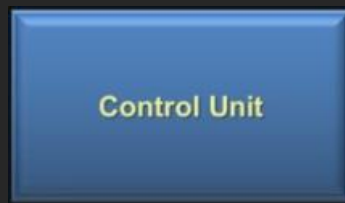
အထက်မှာ Load Instruction ကိုသုံးပြီး 9 ကို CPU ထဲကို ဝန်တင်လိုက်ပါတယ်။ Programmer က အဲ့ဒီ Number ကို CPU ထဲမှာ ထားပြီး Program အသုံးပြုသူကို နံပါတ်ဘယ်လောက်လဲ ခန့်မှန်းခိုင်းပါမယ်။ ဒါဆို Program အသုံးပြုသူက ခန့်မှန်းပြီး သူထင်တဲ့ ဂဏန်းတန်ဖိုးကိုရိုက်ထည့် လိမ့်မယ်။ အဲ့ဒီ အသုံးပြုသူ ရိုက်ထည့်တဲ့ တန်ဖိုး ကို IN instruction နဲ့ ဖမ်းမယ်။ Input Device က Keyboard ဖြစ်တဲ့ အတွက် Keyboard Address အတိုင်းဖမ်းမယ်။ ပီးရင် CPU ထဲမှာ Hold လုပ်ထားတဲ့ နံပါတ်နဲ့ Compare Instruction သုံးပြီး နိုင်းယှဉ်မယ်။ မှန်ရင် JUMP IF = Instruction ကိုသုံးပြီး အောက်က Memory Address ကိုသွားခိုင်းမယ်။ မှားခဲ့ရင် မှားခဲ့ကြောင်း OUT Instruction ကိုသုံးပြီး Output Device ဖြစ်တဲ့ Monitor ကို Letter "G" ကို Output ထုတ်ပေးမယ်။ ပီးရင် JUMP Instruction ကိုသုံးပြီး အထက်က In Instruction ကို ပြန်သွားပြီး နောက်တစ်ကြိမ် ထပ်ကြိုးစားခိုင်းမယ်။ မမှန်မခြင်း ဒီ Process ကို ဆက်တိုက် ဖြစ်နေအောင် Program ကိုရေးသားထားနိုင်ပါတယ်။

သေချာလေ့လာကြည့်ပါ။ Instructions တွေ အားလုံးကိုသေချာ အသုံးပြုသွားတာပါ။ Ourput နဲ့ Input Devices တွေအပါအဝင် JUMF IF Instruction ကိုသုံးပြီး မှန်ရင် Memory Address တစ်ခုကို Randomly သွားလိုက်တာကိုလဲ တွေ့ရမှာပါ။ (နားလည်အောင် သေချာ ဖတ်ပါ)။

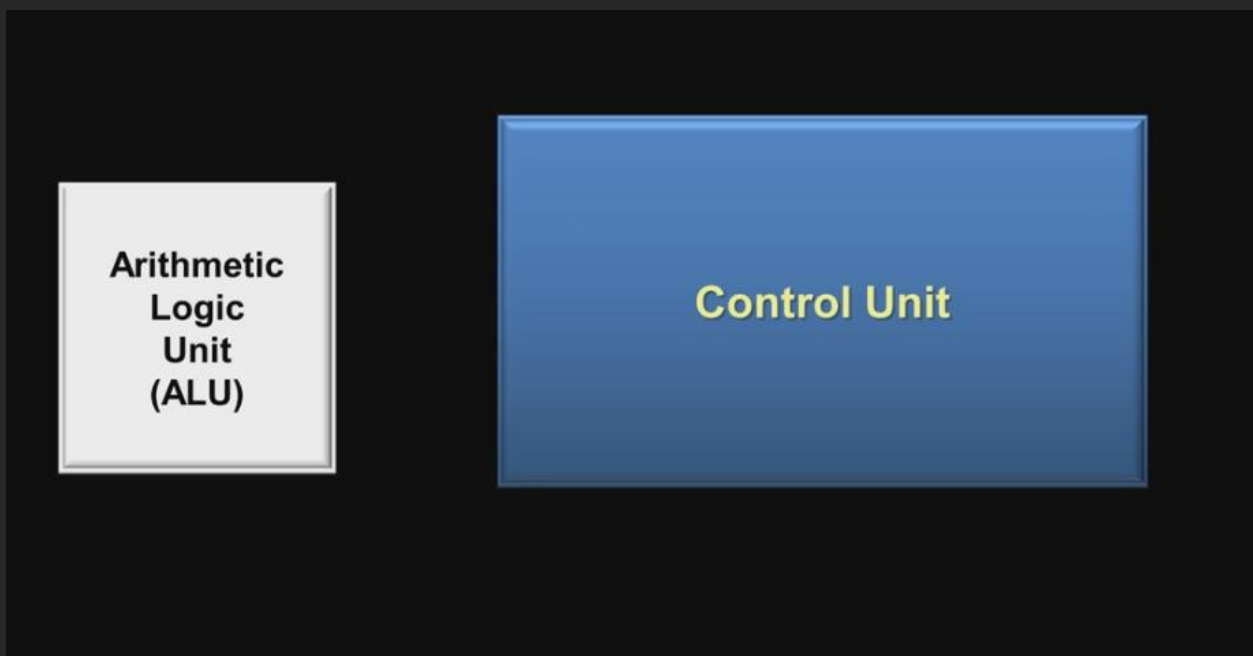
ဒီကိစ္စ RAM ကလွှဲပြီးရင် RAM ကပေးတဲ့ Instruction တစ်ခုရတိုင်းရတိုင်း CPU က Instruction တွေကို ဘယ်လို အလုပ်လုပ်လဲကြည့်ရအောင်

သုံးထားတဲ့ CPU 65-02 ရဲ့ Wiring နဲ့ Scoot CPU တို့ကို သေချာ လေ့လာကြည့်ရမှာပါ။ (အထက်ကမှာ CPU ဖွင့်ထားတဲ့ပုံကိုပြခဲ့ပြီးဖြစ်တဲ့ အတွက် နောက်တစ်ကြိမ်မပြတော့ဘူး)။

CPU မှာ Components တွေအများကြီးပါတယ်။ အရင်ဆုံး စတင်လေ့လာသွားမှာက Control Unit ဆိုတဲ့ Component ပါ။ သူကတော့ တပ်တစ်ခုက အရာရှိနဲ့တူပါတယ်။

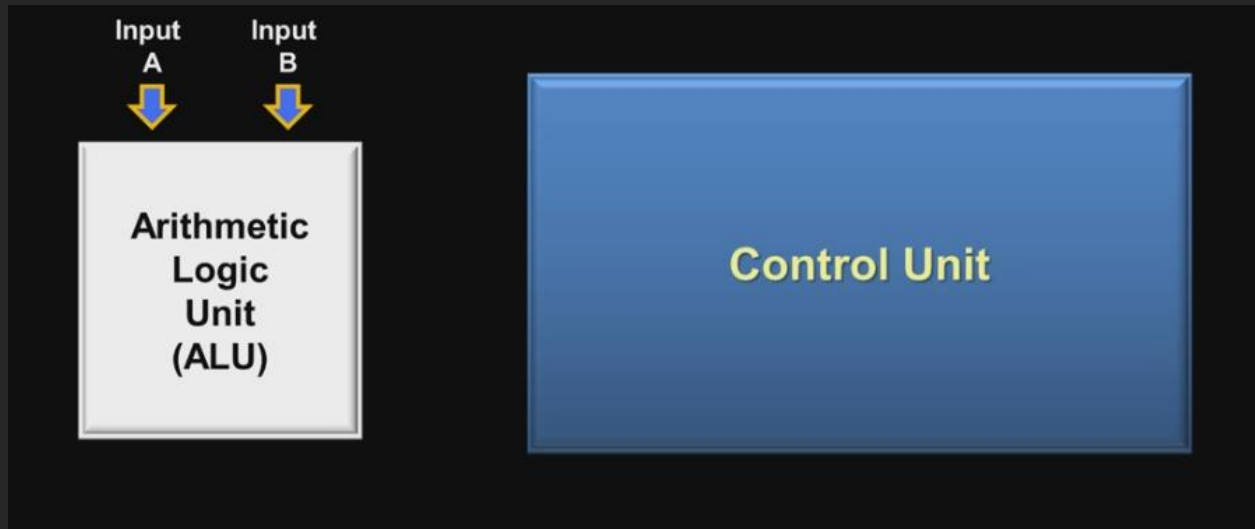


Control Unit က RAM ကနေ ပီးတော့ အမိန့်ကို Instruction တစ်ခု အနေနဲ့ လက်ခံပါတယ်။ လက်ခံရရှိလာတဲ့ Instruction ကို အခြား Components တွေအတွက် သက်ဆိုင်ရာ Commands တွေ အလုပ်တွေအလုပ်ကို ခွဲခြမ်းစိတ်ဖြာလိုက်ပါတယ်။ Control Unit အောက်မှာရှိတဲ့ အရေးအကြီးဆုံး Command တစ်ခုကတော့ Arithmetic Logic Unit ဖြစ်ပါတယ်။ အတိုကောက် ALU လို့ခေါ်ပါတယ်။

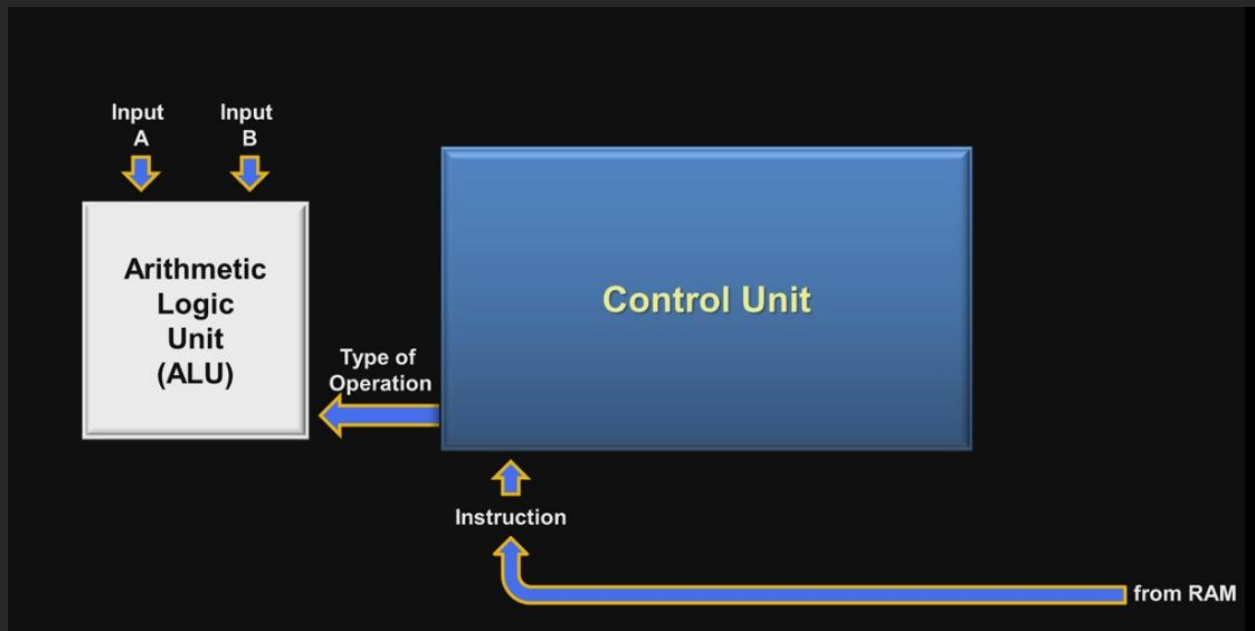


ALU လို့လဲအတိုကောက် ခေါ်တဲ့ Arithmetic Logic Unit ကတော့ CPU ထဲမှာ Arithmetic နဲ့ ပက်သတ်သမျှ အလုပ်တွေ ဖြစ်တဲ့ Adding(ပေါင်းခြင်း) Subtraction (နှုတ်ခြင်း) Comparing (နှိုင်းယှဉ်ခြင်း) တွေအားလုံးကို လုပ်နိုင်ပါတယ်။

ALU မှာ Input နှစ်ခုရှိပါတယ်။ Input A နဲ့ Input B လို့ပဲ ဆိုကြပါစို့။

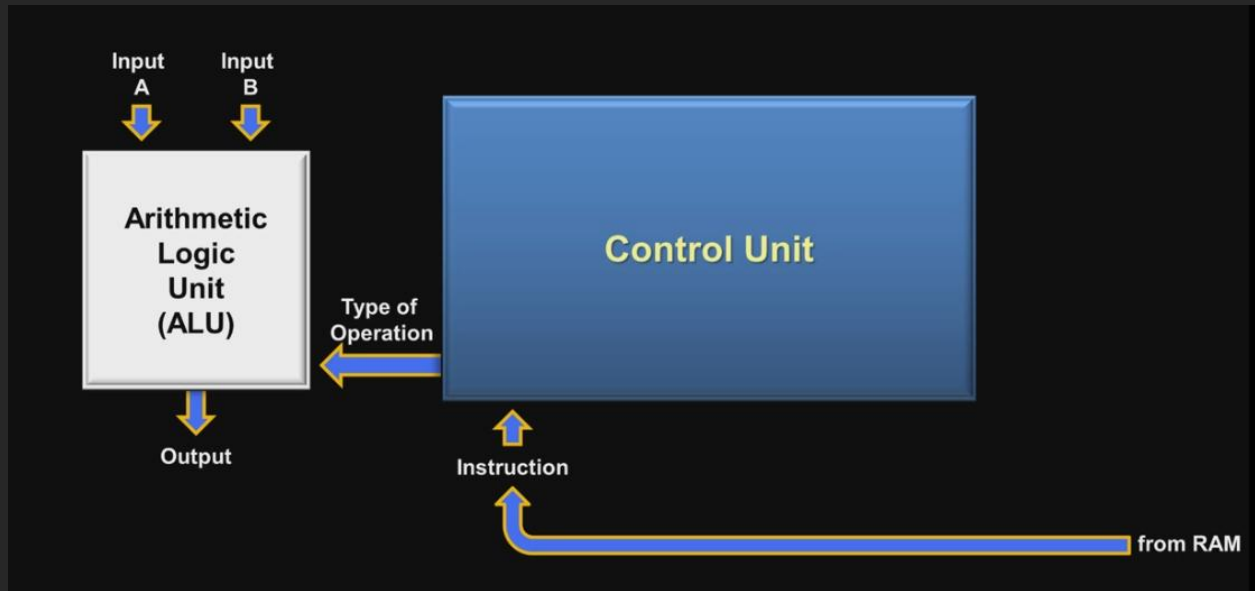


ဆိုကြပါစို့။ ရှေ့က ပန်တင်ခဲ့တဲ့ညွှန်ကြားချက်မှာ ဂဏန်း နှစ်ခုပါတယ်ဆိုပါစို့။ အဲ့ဒီ ဂဏန်း နှစ်ခုကိုလဲပေါင်းချင်တယ်။ ဒါဆိုရင် အဲ့ဒီ ဂဏန်း နှစ်ခုကိုပေါင်းမယ့် ညွှန်ကြားချက်ကို Control Unit က RAM ကနေ လက်ခံရရှိလာပြီး ALU ကို ဘယ်လို အလုပ်ကိုလုပ်ဆောင်ရမလဲ ဆိုတဲ့ ညွှန်ကြားချက်ကို ပို့ပေးတယ်။





ALU က လဲ Control Unit ကနေ လက်ခံရရှိတဲ့ ညွှန်ကြားချက်ကအတိုင်း အလုပ်လုပ်ပီး အဖြေကို Output လုပ်ပေးပါတယ်။



တစ်ခါတစ်ရံမှာတော့ရရှိလာတဲ့ညွှန်ကြားချက်ပေါ်မူတည်ပီး ALU ရဲ့ Output ကို ဘာမှဆက်မလုပ်ပဲ ဥပက္ခာပြုထားရတာရှိပါတယ်။

မနက်ဖြန်ဆက်ရေးမယ် ☹️

ရေးထားတာတွေထဲမှာ အမှား ပါတာတွေရင် ကျေးဇူး ပြုပြီး [brightermyanmar@gmail.com](mailto:brightermyanmar@gmail.com) ကို အကြောင်းကြားပေးပါခင်ဗျာ။

Waiferkolar ရေးထားတဲ့ Android Hacking စာအုပ်ပါ။ Android Hack နဲ့ ပါတ်သတ်သမျှစုံလင်စွာ ဖတ်ရမှာဖြစ်သလို လက်တွေ့ သင်ခန်းစာတွေဖြစ်တဲ့ အတွက် Android နဲ့ ပတ်သက်သမျှ အားလုံး စုံလင်စွာလေ့လာနိုင်မှာဖြစ်သလို လက်တွေ့လုပ်နိုင်သွားမှာဖြစ်ပါတယ်။



BOOK + Applicatoin = 7000 Kyats



