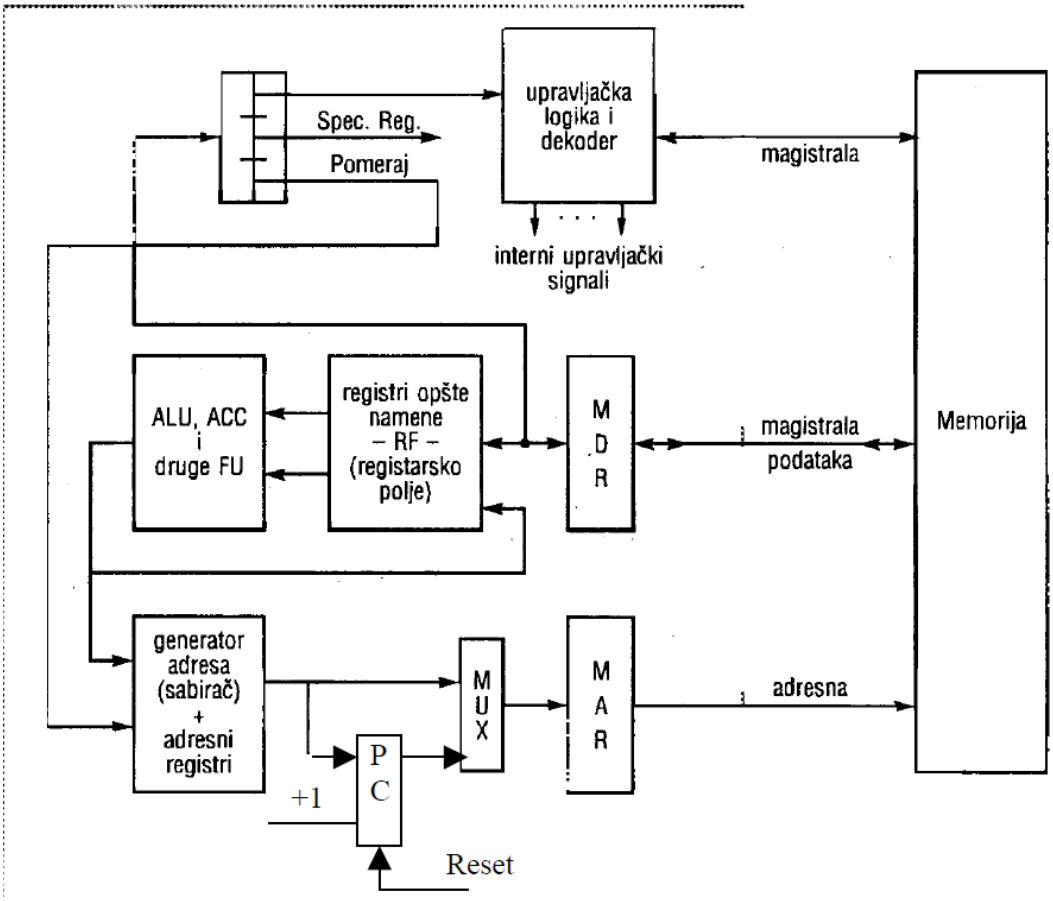
Најчешћа питања из предмета Архитектура Рачунара:

1. **Представите дијаграмом блок шему једноставне CPJ, и опишите рад тог процесора - начин како се извршавају инструкције?**
2. **Функције и врсте управљачких јединица процесора. Управљачке јединице са директним управљањем на бази бројачке секвенце.**
3. **Опишите прекидну технику која користи ланчање и векторско прекидање. Шта се дешава приликом препознавања и признавања прекида, како се обезбјеђује улазак и излазак из прекидне сервисне рутине? Које су функције прекидне сервисне рутине?**
4. **Својства хијерархијске организације меморије. Мапирање виртуелнх у физичке адресе инвертованим страничењем?**
5. **ДМА техника У/И комуникације.**
6. **Мапирање виртуелних адреса у физичке страничењем.**
7. **У/И портови: врсте и улога . Начин селекције приступа У/И уређајима.**
8. **Опишите прекидну технику У/И комуникације. Ко и на који начин иницира комуникацију, шта се дешава током иницирања, прихватања, идентификације и послуживања прекида, како се остварује улазак и излазак из прекидне сервисне рутине? Које су функције прекидне сервисне рутине?**
9. **Ланчање и векторско прекидање. Идентификација сервиса на основу вектора прекида. Логика блока ланчања.**
10. **Концепт микропрограмске управљачке јединице. Представите блок шему микропрограмске управљачке јединице и опишите како микропрограмска управљачка јединица генерише управљачке сигнале приликом извршења машинских инструкција.**
11. **Опишите и представите дијаграмом активности које се извршавају у процесору у току циклуса извршавања инструкције.**
12. **Програмирани У/И. У којим ситуацијама је ова техника У/И комуникације прихватљива, а када је потребно користити друге технике ? Како је могуће реализовати селекцију У/И уређаја и приступ подацима на тим уређајима ?**
13. **Ланчање и векторско прекидање. Опишите како се обезбјеђује усмјеравање програмске секвенце на рутину за сервисирање прекида (овом техником) и шта се дешава приликом уласка у прекидну сервисну рутину.**
14. **Представите дијаграмом различите начине адресирања и укратко их опишите.**
15. **Наведити и опишите концепт пројектовања паралелних програма. Шта је најчешћи циљ који се жели остварити при паралелном извршењу програма, и који су (међусобно конфликтни) принципи којима се жели остварити жељени циљ ? Шта је резултат примјене претходних принципа?**
16. **Спрежна мрежа FNN (решетка).**
17. **Асоцијативни процесори.**

18. **Узроци временских губитака у паралелним системима и њихов утицај на концепт развоја паралелних програма. Опишите фазе развоја паралелних програма.**
19. **Систолични процесори.**
20. **Хардверски базирана спекулација. Улога ROB-а (Reorder Buffer).**
21. **Спрежне мрежа PM2I.**
22. **Генерализована коцка.**
23. **Зашто је важно предвиђање гранања ? Опишите основне технике које се користе за предвиђање гранања.**
24. **Објасните контролну зависност између инструкција. Да ли је очување контролне зависности при извршењу инструкција неопходан услов за коректно извршавање програма? Шта се мора очувати приликом (паралелног) извршавања програма као би се обезбиједила коректност извршења ?**
25. **Функције управљачке јединице процесора.Управљачке јединице са директним управљањем.**
26. **Хиперкоцка**
27. **Приказати дијаграмом управљачке сигнале које користи и продукује управљачка јединица процесора.**
28. **Опишите концепт SIMD архитектура и наведите врсте SIMD процесора.**

Одговори на питања:

1. Једноставна блок шема ЦПЈ:



Основна функција ЦПЈ јединице је да извршава програм смјештен у главној меморији.Састоји се од двије основне функционалне јединице,а то су : УПРАВЉАЧКА и ИЗВРШНА јединица.Управљачка једница врши дохватање инстуркција из главне меморије,затим декодовање тих инструкција и потом генерисање одговарајућих управљачких сигнала за извршну једницу.Извршна једница извршава операције предвиђене датом инстуркијом.Подаци унутар ЦПЈ циркулишу у оквиру једне или више магистрала, те магистрале су познате под именом интерне магистарале.

2. Функција управљачке једнице процесора је да генерише сигнале за управљање свим активностима унутар ЦПЈ, контролом управљачке јединице се врши дохватање инструкције из меморије,њено декодовање и идентификација и пренос операнада од изворишта до функционалних једница у којима се обавља дата операција, те враћање резултата на одредиште. Овај процес се под контролом управљачке јединице понавља при извођену сваке следеће инструкције.Управљачка једница овде дјелује као неки командни центар из којег се управља радом осталих јединица система,а све у сврху извршавања машинских инстуркција.Да би управљачка јединица остварила све наведене задатке,она мора да:
   - обезбједи коректан редосљед извршења инструкција
   - генерисањем одговарајућих сигнала обезбједи извршавање селектоване функције

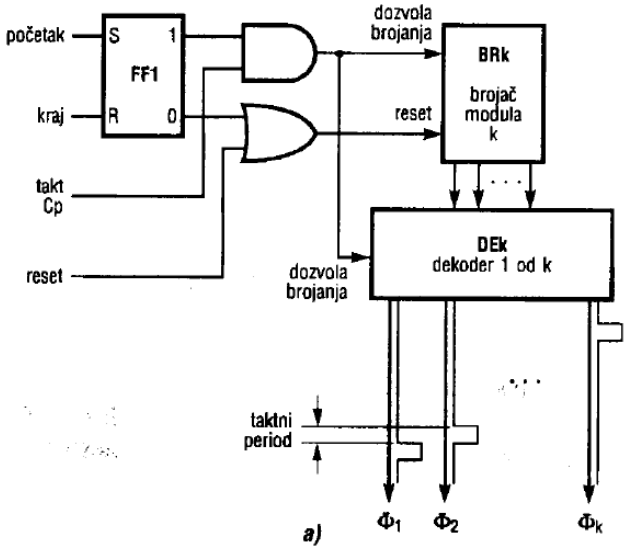
   Што се тиче врста управљачких једница генерално их можемо класификовати у две врсте :

   ❖ управљачке јединице са директним управљањем – логичком мрежом се генеришу контролни сигнали у предефинисаном редосљеду за сваку инструкцију. Даље ове управљачке једнице се дјеле на основу реализације на :
   - ➢ на оне засноване на табелама стања
   - ➢ засноване на елементима за кашњење
   - ➢ засноване на бројачкој секвенци

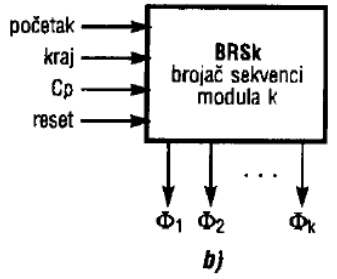
   ❖ управљачке једнице са микропрограмским управљањем – контролни сигнали се генеришу извршењем микроинструкција записаних у интерној микропограмској меморији процесора,а извршење сваке машинске инструкције се реализује низом елементарних акција које се имплементирају микроинструкцијама.Микроинструкције могу да буду хоризонталне – њима се спецификује извршавање више микрооперација и вертикалне – ако се њима спецификује извршавање једне микрооперације.

   Закључујемо да је предност директног управљања брзина, а микропрограмског једноставност реализације и измјена/отклањања грешака у пројектовању.

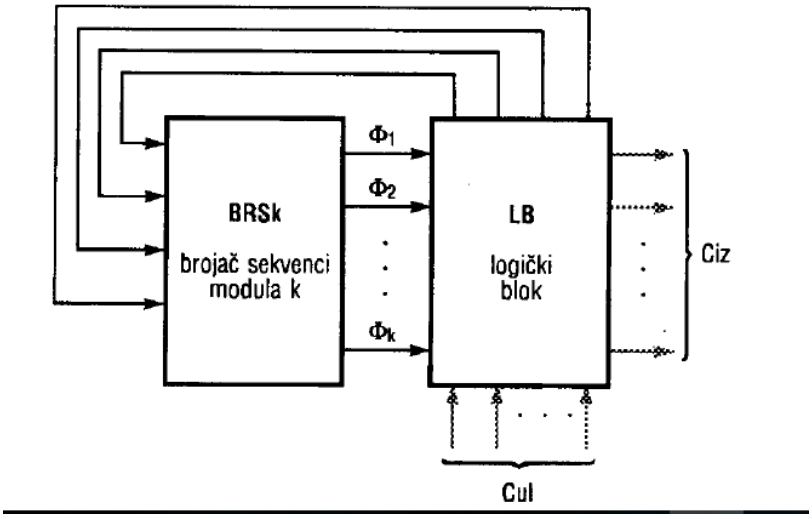
   Управљачка једница реализована преко бројачке секвенце – управљачке једнице овог типа претпоставља извршење инструкције у к корака.У и-том кораку се активира неки скуп управљачких линија коришћењем односног фазног импулса $\Phi_и$.Фазни импулси генеришу се колом бројач секвенци .Сукцесивни импулси на излазу бројача су помјерени за време трајања импулса.



Спрега бројача секвенци и логичког блока којим се имплемнтира односна функција је предестављена на слици:



Sl.3.9 Brojač sekvenci modula *k*



4. Основна својства хијерархијске организације меморије :
   - Инклузија
   - Кохеренција
   - Локалност

   Особине инклузије се исказује са :

   $M_1 \subset M_2 \subset M_3 \subset \ldots \subset M_n$.

   Инклузија скупова имплицира да су све информације оригинално похрањене у спољнем нивоу M_n. У току обраде, дио информација се копира у M_{n-1}. Аналогно, подскуп у M_{n-1} се копира у у M_{n-2} итд. Тј. ако се информациона ријеч налази у у M_i , онда се копије те информације такође налазе у свим вишим нивоима у M_{i+1} ,M_{i+2}, … , M_n ,али информација која се налази у M_{i+1} не мора постојати у M_i у посматраном тренутку. Непостојање речи у M_i имплицира непостојање ријечи у свим нижим нивоима (M_{i-1}, M_{i-2}, … , M_1).Највиши ниво је меморија за архивирање где се могу наћи све информације.

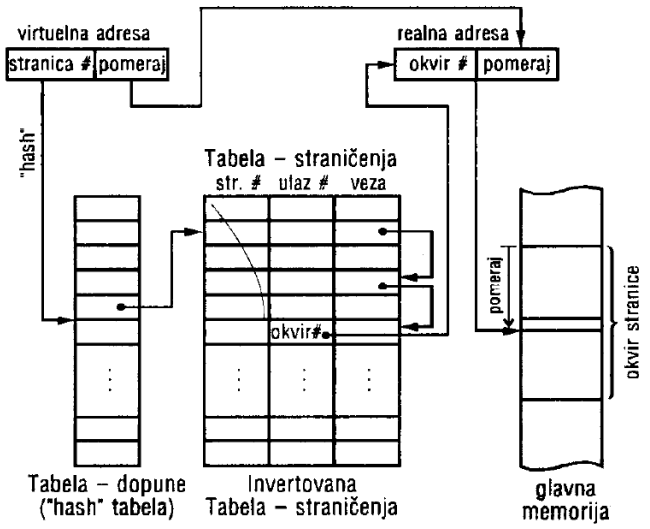
Својство кохерентности захтјева да информација на неком нивоу,има конзистентне копије на следећим вишим нивоима . Одржавање кохерентности се може реализовати следећим стратегијама :

- упис одмах (WT – write through)
- одгођени упис(WB – write back)

WT стратегија ажурира одговарајућу копију у M_{i+1}, чим је дошло до ажурирања неке речи у M_i .. WB метод одгађа ажурирање у M_{i+1} ,све до момента док се односна информација не уклања из M_i .

Локалност референцирања је својство програма да меморијске адресе, генерисане у току извршавања програма, имају тенденцију груписања посматрано са аспекта времена и простора. У највећем броју програма 90% времена програмског извршења проистиче извршавањем 10% програмског кода. Као посљедицу имамо својство локалности : ово својство има три димензије : темпоралну,просторну и секвенцијалну. Темпорална локалност је својство да ће меморијске референце,генерисане у најближој прошлости,вјероватно поново бити генерисане у скоријој будућности. Просторна локалност је својство да се генеришу меморијске референце које су блиске једна другој. Секвенцијална локалност се односи на чињеницу да се често генерише секвенцијални низ адреса.Перфорамнсе меморијског система зависе од ефективног времена приступа било ком нивоу у меморијској хијерархији и од фреквенције приступа тим нивоима.

У случају великог виртуелног адресног простора ПТ би постале изузетно велике. У таквим случајевима користе се инвертоване табеле страничења (ИПТ). Код ове технике инвертована табела страничења (ИПТ) садржи један улаз за сваку физичку страницу (фраме). Према томе, број улаза је одређен бројем физичких умјесто виртуелних страница. Поље веза формира ланац ИПТ улаза који имају исту хаш вриједност. Нпр, хаш функција може мапирати број виртуелне странице → индекс улаза у ИПТ на слиједећи начин:

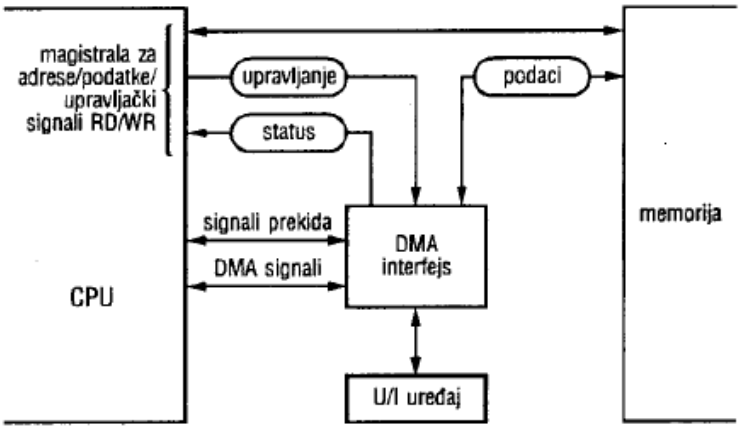


индекс улаза у ИПТ = број виртуелне странице мод н гдје је н број физичких страница (фраме-ова). Садржај вриједности виртуелне адресе страница# (вриједност кључа) се упоређује са вриједношћу стр.# у улазу у табели страничења који одговара вриједности хаш функције за дату виртуелну страницу. Ако су вриједности исте, онда се одговарајући број физичке странице добија у пољу улаз#. Ако се вриједности кључа не поклапају, онда се преко поља 'веза' иде на слиједећи улаз са истом хеш вриједношћу и поступак понавља док се не пронађе физичка страница, или утврди да текућој виртуелној адреси није

придружена ниједна (физичка) страница у главној меморији (фраме). У посљедњем случају генерише се странична грешка.

5. Комуникација између периферних јединица и процесора је релативно једноставан и ефикасан механизам, уколико су периферијске јединице споре. У том случају, и за велики број периферијских јединица , вријеме које процесор потроши на комуникацију са периферијама износи свега неколико процената укупног процесорског времена. Међутим трансфер великог броја података између брзих периферија и меморије преко процесора механизмом прекида за трансфер сваког појединачног податка захтјевао би знатан проценат процесорског времена. Према томе као решење се намеће ДМА приступ.

   Канал за директан приступ меморији је специјални интерфејс који омогућава периферији да изврши брз трансфер података ка/из меморије ,без учешћа ЦПЈ. Шема ДМА У/И комуникације дата је на слици :



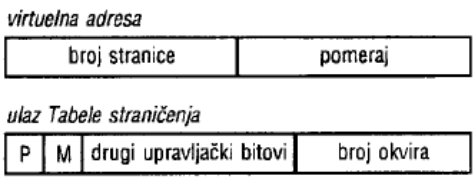
   Да би ДМА извршио пренос података ка меморији, он стандардно поставља процесору захтјев за прекид INTR. Процесор поставља параметре преноса : адресу почетка меморијског блока , број блока диска, као и број бајта који се преноси, те издаје команду за пренос која садржи и информацију о смјеру преноса. ДМА интерфејс треба да добије право управљања магистралом у току преноса података ка/из меморије. Захтјев за управљање магистралом се шаље преко линије BR(bus request), а одобрење се добија кад процесор постави сигнал BG.Прије предавања магистрале ДМА интерфејсу, процесор треба да заврши текуће коришћење магистрале и да постави своје излазе на магистралу на ниво високе импедансе.Постоји неколико варијанти ДМА преноса:

   - Крађа циклуса – процесор не троши сво своје време на комуникацију са меморијом. ДМА постаје контролер над сабирницом ДМА када је у процесу трансфера података и када процесор не користи магистралу (ДМА краде циклусе контроле над магистралом података)
   - Стандардни ДМА пренос. Код ове варијанте ДМА интерфејс држо контролу над сабирницом за вријеме преноса цијелог блока података. Овим се постижу велике брзине преноса, али процесор мора да чека док ДМА не заврши трансфер блока. Компромисно рјешење је да ДМА контролер

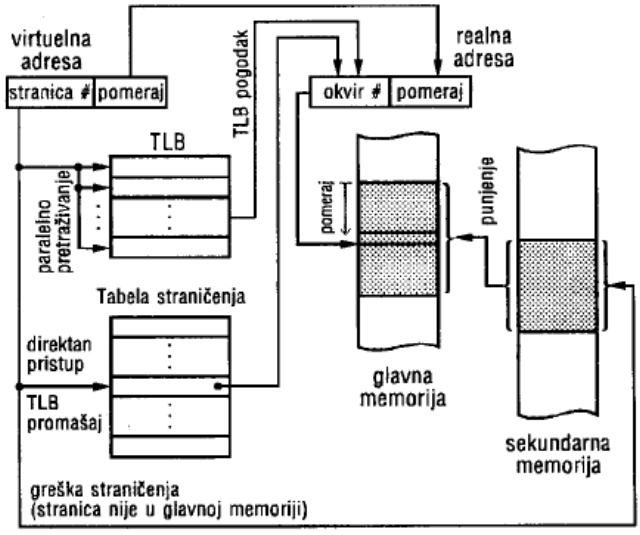
ослобађа сабирницу након преноса одређеног броја подблокова, како би процесору омогућио да реагује на изузетне догађаја

- По трећој варијанти ДМА контролер користи магистралу у машинским циклусима у којима је не користи ЦПУ. На тај начин се максимизира искоришћење пропусног опсега меморије и то без блокаде рада ЦПУ.

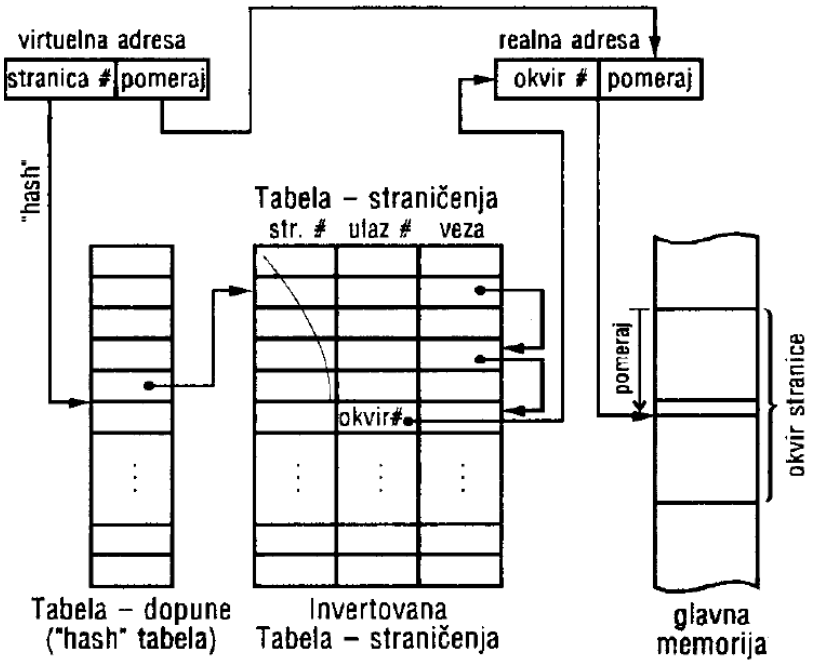
6. Код технике страничења, у циљу пресликавања виртуелног у физички адресни простор и виртуелни и физички адресни простор се дијеле на странице исте дужине. Постоје механизми транслација који користе транслационе мапе које се могу имплементирати на различите начине. Ове мапе могу бити смјештене у кеш меморију, специјалну асоцијативну меморију или у главну меморију. Свака мапа садржи одређени број улаза, а сваким улазом се спецификује транслација. Приступ одговарајућем улазу у табели се врши на бази функције мапирања,која се примјењује на виртуелну адресу. Мапирање се може имплементирати хаш или конгруентном функцијом.



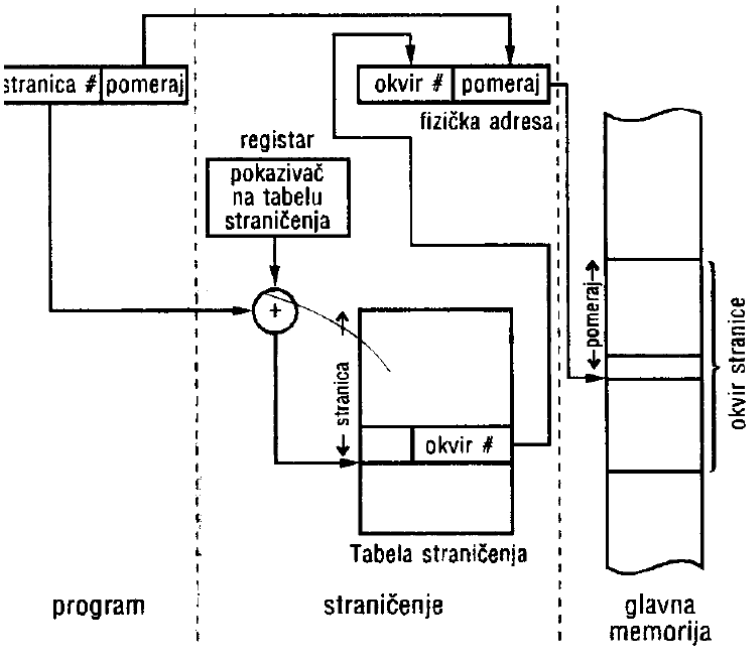
Варијанте транслационих мапа су кеш страничења (ТЛБ – табле лоокасиде буффер) и табела страничења (ПТ – паге табле).ТЛБ је брза меморија која садржи најчешће референциране улазе табеле страничења. Најповољнија варијанта је ако се транслација виртуелне у физичку адресу реализује преко ТЛБ улаза. Уколико ТЛБ не садржи улаз који спецификује транслацију, онда се тражење наставља у табели страничења. Хадрверска организација система страничења која користи ТЛБ дата је на следећој слици:Табеле страничења (ПТ) садрже суштински исте улазе као и ТЛБ (који спецификују асоцијацију између виртуелних и физичких страница – оквира). Ове табеле се креирају у главној меморији након креирања корисничких процеса (за сваки процес се креира посебна ПТ). С обзиром да број апликативних процеса може бити велики, број ПТ у меморији може такође бити велики. И ТЛБ и ПТ се морају динамички ажурирати како би одражавали актуелно стање. Мапе транслације представљају текући снимак стања меморијског референцирања. Уколико се тражена страница не налази у меморији (не постоји пар (виртуелна страница, физичка страница) у ПТ), онда се генерише странична грешка (паге фаулт). У том случају се текући процес суспендује, активира се нови процес (извршава се цонтехт свитч), а тражена страница се добавља са диска/јединице магнетне траке, у меморију. Након тог трансфера суспендовани процес се ставља у листу спремних процеса, и након његовог активирања, наставља се извршење од оне инструкције која је генерисала страничну грешку. ПТ се могу



имплементирати и у више нивоа. Овим се де-фацто формира стабло за превођење адреса, са могућношћу проширења меморијског простора који се мапира и имплементацијом софистификоване заштите приступа страницама. Примјер превођења виртуелних у физичке адресе код технике страничења, дат је на слици изнад.У наведеном примјеру, у посебном регистру се чува табела страничења процеса.



Број виртуелнестранице се користи као индекс у тебели страничења, из које се чита одговарајући број физичке странице (број оквира). Овај број се комбинује са помјерајем виртуелне адресе чиме се добија жељена реална (физичка) адреса. Претходно описана техника (директног) страничења је погодна ако виртулни адресни простор није изузетно велики (до 32 бита). У случају великог виртуелног адресног простора (52 битне виртуелне адресе код ИБМ PC/6000, нпр) ПТ би постале изузетно велике. У таквим случајевима користе се инвертоване табеле страничења (ИПТ). Код ове технике инвертована табела страничења (ИПТ) садржи један улаз за сваку физичку страницу (фраме). Према томе, број улаза је одређен



бројем физичких умјесто виртуелних страница. Поље веза формира ланац ИПТ улаза који имају исту хаш вриједност. Нпр, хаш функција може мапирати број виртуелне странице → индекс улаза у ИПТ на слиједећи начин: индекс улаза у ИПТ = број виртуелне странице мод н гдје је н број физичких страница (фраме-ова). Садржај вриједности виртуелне адресе

страница# (вриједност кључа) се упоређује са вриједношћу стр.# у улазу у табели страничења који одговара вриједности хаш функције за дату виртуелну страницу. Ако су вриједности исте, онда се одговарајући број физичке странице добија у пољу улаз#. Ако се вриједности кључа не поклапају, онда се преко поља 'веза' иде на слиједећи улаз са истом хеш вриједношћу и поступак понавља док се не пронађе физичка страница, или утврди да текућој виртуелној адреси није придружена ниједна (физичка) страница у главној меморији (фраме). У посљедњем случају генерише се странична грешка.

7. Прво ћемо да кажемо да порт није ништа друго него један У/И регистар. Периферијски интерфејс садржи скуп регистара – У/И портова којима CPU може да приступа и преко којих CPU комуницира са интерфејсом. Функције портова су:
   - баферовање података ка/из меморије
   - чување информације о статусу уређаја са којим се комуницира
   - регистровање команди CPU-а упућених У/И интерфејсу

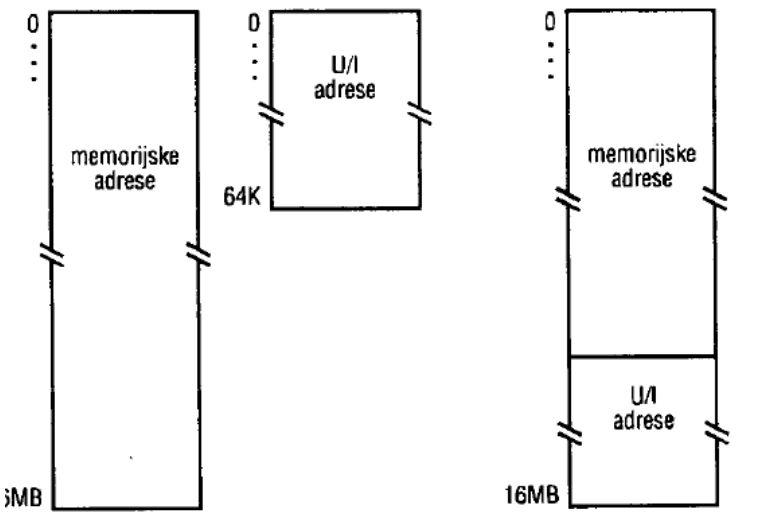
   Скуп портова представља У/И програмски модел.

   Портове можемо да поделимо према типу података који се размењују између порта и процесора, тиме добијамо следећу поделу:
   - порт података – преко којег се размењују подаци
   - статусни порт – преко којег процесор може добити информације о стању преноса,У/И интерфејса односно уређаја
   - управљачки порт – преко којег CPU поставља команде и параметре за извођење У/И операције

   Поред ове имамо и просту подјелу на улазне и излазне портове.Разлика између ова два порта је у томе што улазни порт не памти садржај него га само баферује за прослеђивање на У/И магистралу,док је излазни порт регистар са меморијом који памте вредност све док CPU не упише нову.

   За селекцију У/И уређаја и трансфер података ка периферним уређајима (и обрнуто),процесор може да користи један од два начина.Први начин се изводи коришћењем специјалних У/И инструкција процесора , познат је и као изолирани И/О ,док се други начин реализује коришћењем инструкција за измену података са меморијом – Меморијско-мапирани У/И (даље у тексту ММУ/И).

   IN/OUT инструкције извршавају трансфер података слично меморијским инструкцијама LD/ST , изузев да не адресирају меморију него У/И порт. При извођењу меморијских инструкција процесор активира сигнал М/ИО за селекцију меморије (логичка 1) . Сигнал М/ИО се користе како за селекцију У/И интерфејса,тако и за иницијализацију самог трансфера.При трансферу података иницираног са IN/OUT инстукцијама меморија није прозвана јер је контролни сигнал М/ИО постављен на 0. На тај начин адресе селектују или И/О портове или меморијске локације зависно од коришћених инструкција. Код меморијски мапираног У/И, И/О портовима се приступа коришћењем меморијских инструкција. Један дио меморијског адресног простора је придјељен за адресирање У/И портова.



a) izdvojeni U/I prostor *b) memorijsko preslikani U/I prostor*

Предности ММУ/И комуникације су:

- ✓ не захтјева специјализоване У/И инструкције
- ✓ за У/И трансфер се могу користити све инструкције за референцирање меморије
- ✓ расположив је велики број портова
- ✓ поједностављује се хардверска структура

Мане ММУ/И комуникације су:

- губитак дјела адресног простора
- меморијске инструкције нису оптимизоване за У/И операције

8,3. Прекиди су догађаји који узрокују прекид извршавања нормалног тока програмске секвенце. Идеја прекида потиче од проблема комуникације са периферијама, Код програмираног У/И процесор континуално тестира спремност пероферије за комуникацију.При таквом начину комуникације, у систему са много периферних једница процесор би већину времена проводиу у чекању на спремност периферија. Такав начин рада би успоравао чак и периферије : периферија која има спреман податак би морала да чека док процесор заврши комуникацију са неком другом периферијом.Алтернативни приступ је да процесор ради користан посао изводећи основни програм,а да периферија кад постане спремна за комуникацију о томе обавјести процесор – тј. постави захтјев за пренос. На основу приоритета захтјева, процесор може прихватити захтјев и сервисирати догађај активирањем прекидне рутине, или одложити сервисирање захтјева за неки каснији тренутак времена.Процесор може прекинути текућу секвенцу само између индивидуалних машинских инструкција. Разлог томе је што програм који се извршава треба да настави од тачке у којој је прекинут, након сервисирања догађаја који је послао захтјев за прекид.У суштини процес извршавања једне инструкције се може описати секвенцом :

```
BEGIN
        CheckForInterrupt;
        Fetch;
```

Execute;

END

CheckForInterrupt секвенца утврђује постојање захтјева за прекид, односно догађаја који захтјева послуживање. Уколико се ради о догађају који зајтјева неодложно сервисирање или ако је прекид који се може контролисати дозвољен, онда се улази у процес услуживања унутар којег се :

- похрањује у меморију дио стања или комплетно стање програмског модела процесора и поставља се у ПЦ адреса почетка рутине за сервисирање прекида
- идентификује се извор и изводе се све потребне акције за сервисирање прекида
- сачувано стање програмског модела се рестаурира са стека и наставља се извођење од мјеста прекида

Овако у суштини изгледа комуникација са У/И коришћењем прекида. Што се ове комуникације тиче постоје три варијанте коришћења прекида у процесу У/И комуникације. Прво имамо системе са већим бројем прекидних линија:

Најједноставнији начин реализације У/И комуникације прекидним механизмом је да свака периферија има своју прекидну линију. Свака линија којом се захтјева прекид има обично фиксан приритет.Омогућење прекида одређених приоритета се контролише интерним битовима дозволе. За сваку прекидну линију дефинисана је специфична адреса на коју се преусмјерава програмска секвенца у случаја догађаја прекида.У оквиру интерне CheckForInterrupt секвенце процесора се :

- утврђује захтјев за прекид највишег приоритета у ИР
- уколико је прекид тог нивоа дозвољен, онда се улази у секвенцу послуживања прекида.Адреса ИСР је одређена захтјевом INTRи који се послужује

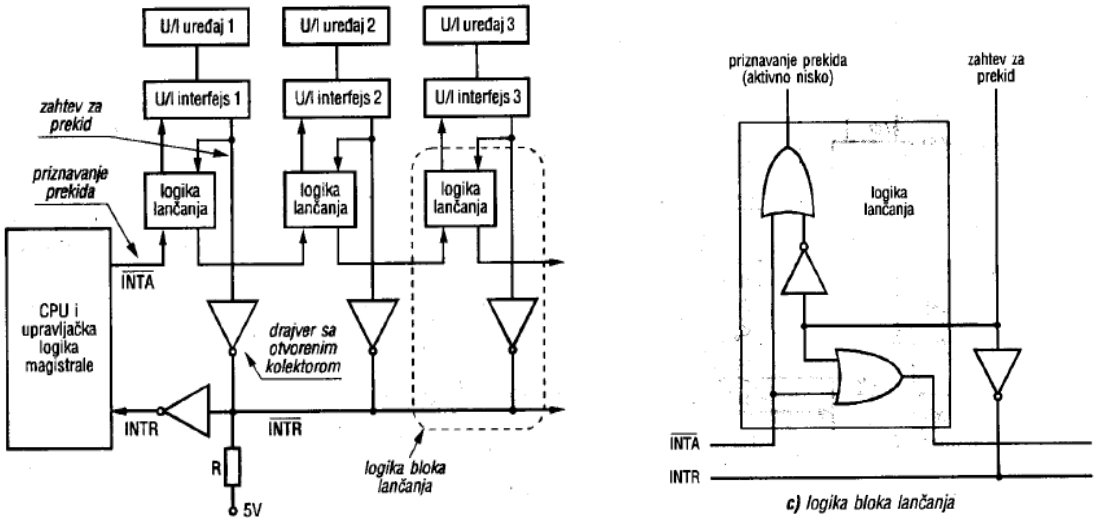
Овај концепт није погодан када имамо већи број интерфејса. Због тога се користи рјешење код којег се на једну линију захтјева веже већи број У/И интерфејса.Тако долазимо до технике идентификације прозивањем. Код ове технике када се деци прекид узрокован захтјевом за прекид INTRи на одређеној линији, улази се у процес послуживања прекида активирањем одговарајуће ИСР-е .При уласку у ИСРи није познат итвој прекида, јер је виште У/И интерфејса везану на исту INTRи линију. Да би се утврдио извор , односна ИСР рутина испитује сваки уређај везан на дату линију. Први интерфејс за који се утврди да је послао прекид опслужује се у ИСРи.Ова варијанта је добра ако је мала вјероватноћа да два интерфејса траже истовремено захтјев за прекид. Даље имам технику ланчања која је објашњена у следећем одговору. Још нам остаје метод векторског прекида заузећем магистрале. Код овог метода У/И интерфејс треба прије постављања захтјева да стекне право контроле над магистралом. На тај начин само један У/И може да активира ову линију. Након активирања захтјева и његовог детектовања од стране процесора, процесор шаље сигнал препознавања интерфејса, а интерфејс поставља свој вектор прекида на линијама података.

9,13. Прво ћемо рећи нешто о прекидима. Прекиди су догађаји који узрокују прекид извршавања нормалног тока програмске секвенце. Ови догађаји могу да буду хардверски

или софтверски. Прекид произилази из вишег приоритета сервисирања догађаја који узрокује прекид, у односу на приоритет извршавања основног програма. Сервисирање се реализује активирањем прекидне рутине која треба да изведе све потребне акције захтјеване појавом датог догађаја. Након сервисирања догађаја прекида програмска секвенца се враћа на тачку прекида , након чега се наставља извођење програма.

Ланчање и векторско прекидање.

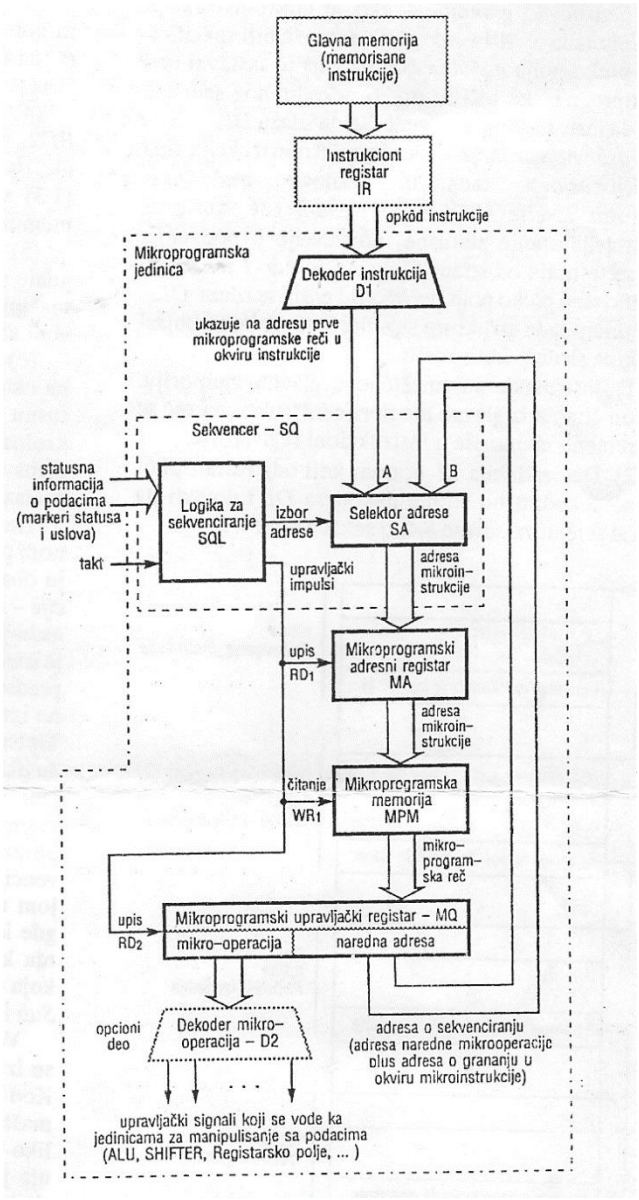
Код ове технике и даље је више У/И интерфејса везано на једну INTRи линију – захтјев за прекид. Идентификација У/И интерфејса који је захтјевао прекид и активација рутине за послуживање односног интерфејса се реализује хардверски. Да би се активирала односна ИСР рутина интерфејса који је поставио захтјев за прекид, односни интерфејс поставља на линије података прекидни вектор (идентификатор на основу којег процес идентификује ИСР). Да би само један уређај од више њих који су евентуално истовремено поставили захтјеве за прекид, поставио прекидни вектор на линију података користи се ланчање.



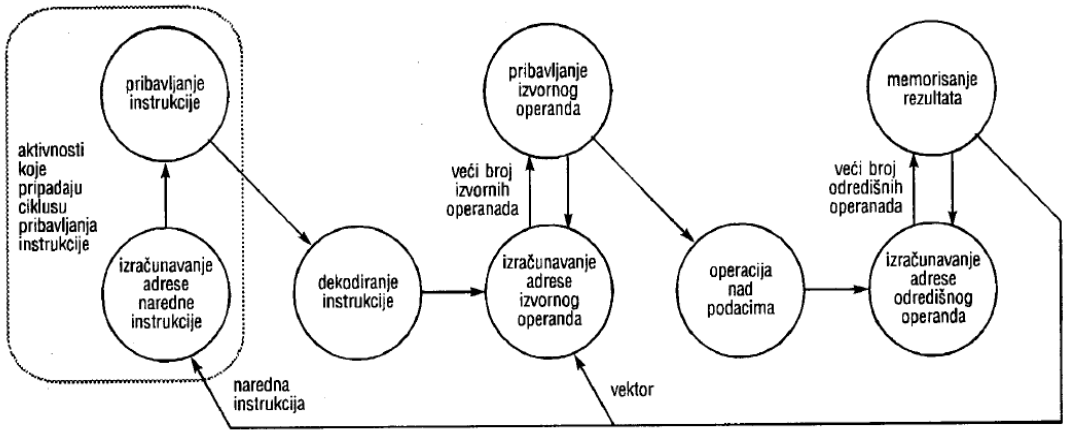
ИНТА се користи као сигнал да је CPU препознао захтјев за прекид и да је спреман да прими информацију о вектору прекида. Овим се остварује синхронизација CPU-а и У/И интерфејса у процесу идентификације захтјева за прекид.

10. Концепт микропрограмске управљачке јединице: Управљачка јединица генерише контролне сигнале извршавањем одговарајућих микроинструкција записаних у интерној микропрограмској меморији процесора. Извођење сваке машинске инструкције се реализује низом елементарних акција које се имплементирају микроинструкцијама. Микроинструкције могу да буду хоризонталне и вертикалне. Структура управљачке јединице са микропрограмским управљањем :
    - декордер функција
    - микропрограмска меморија
    - микропрограмси адресни регистар
    - микропрограмски управљачки регистар
    - секвенцер
        - селектор адресе
        - логика са секвенцирање

    Блок шема микропрограмске управљачке јединице:



11. Активности у току циклуса су приказане следећом сликом:



Slika 3.3 Aktivnosti u toku ciklusa instrukcije

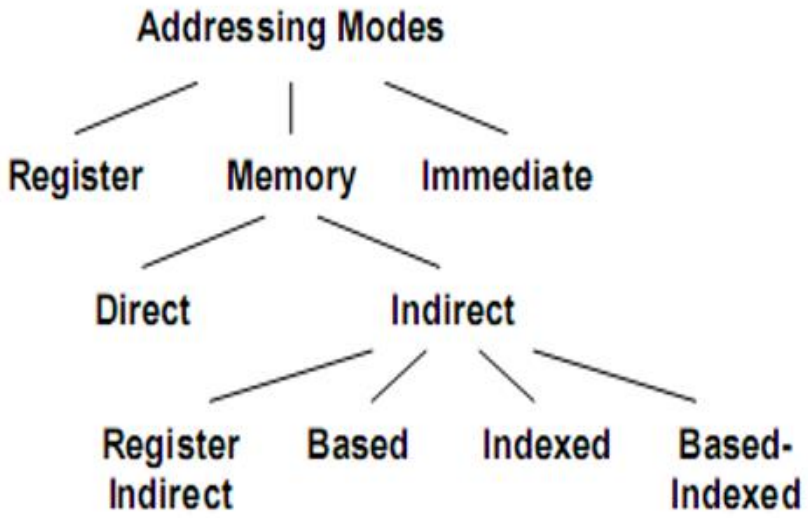
Као што знамо програм који се изводи на рачунари мора да се налази на основу Фон Нојманове шеме у главној меморији.Програм у главној меморији може да буде трајно уписан (РОМ меморија) што се у пракси углавном користи код специјализованих рачунара,најчешћи приступ је да се као главна меморија користи РАМ меморија.Управљачка јединица добавља инстуркције из меморије,декодира их и генерише одговарајуће управљачке сигнале.Извршна јединица извршава команде специфициране датом инструкцијом.Као што видимо, многи резултати извршења се користе као улазни подаци за неке следеће инстуркције. Због тога морамо омогућити чување тих привремених резултата,познатих као међурезултати.Како је сам процесор ограничен по питању меморије јер има ограничен број регистара који су потребни за извршавање неке друге инструкције.Онда се као складиште може користити и главна меморија у том случају у главној меморији се не налазе само програми и инстуркције већ и подаци над којим се врше дате инструкције.Уколико капацитет главне меморије није довољан да се сачувају потребни подаци онда се могу користи јефтина решења као што су магнетни дискови.Само је у овом случају потребно нагласити да процесор не може директно да приступа подацима из секундарне меморије већ он то може само преко главне меморије.

12. При У/И комуникацији CPU може бити у већој или мањој мјери укључен у процес У/И комуникације. Ако су У/И комуникације у потпуности контролисане од стране CPU-а ,онда се овакав тип трансфера назива програмирани У/И. Постоје две варијанте програмираног У/И преноса и то су условни и безусловни У/И пренос. Код условног преноса CPU провјера спремност интерфејса и уколико интерфејс није спреман он га чека све док не постане спреман. У варијанти безусловног преноса процесор не испитује статус спремности периферије за комуникацију, ово се често користи када се зна да је периферија увјек спремна за У/И комуникацију у тренутку иницирања трансфера. Оваква техника преноса је погодна за комуникацију са уређајем који је увјек спреман, у супротном имамо велики недостатак јер имамо огромни губитак процесорског времена док се чека на спремност

периферије , у осталим ситуацијама погодно је користи друге технике У/И комуникације и преноса као што су прекидна и пренос директним присупом меморији.

Селекција уређаја и приступ подацима на уређају се може урадити на два начина тј. постоје генерачно две опције изоловани У/И – код којег са приступ уређају, а самим тим и подацима на уређају користимо посебне инструкције ИН/ОУТ, док код другог приступа познатог као меморијски мапиран У/И – за приступ користимо стандарне инструкције за референцирање меморије, у овом начин губимо одређени дио меморијског адресног простора.

14 .



15. Преструктурирањем постојећих алгоритама или конструкцијом нових, може се обезбиједити знатно повећање износа паралелизма. Конкурентност се појављује као фундаментални захтјев за алгоритме и програме. Даље, многи постојеци секвенцијални језици садрже конструкције које лимитирају могућност детекције паралелизма у постојећим програмима. Секвенцијални програми нису аутоматски прилагодљиви за извођење за случај повећања броја процесора у систему и обима проблема. Могућност проширења у наведеном смислу се такође намеће као један од битних захтјева за паралелне програме. Комплексност паралелног програмирања изразито потенцира модуларност као фундаментални принцип програмирања. Аспекти модуларности секвенцијалног програмирања морају бити просирени, с обзиром на специфицност паралелних система. Оптимално извођење подразумијева постојање функције циља која може бити: минимално вријеме извршења програма (алгоритма), равномјерно оптерећење процесних ресурса, минимална цијена извођења и сл. Пројектовање се мозе структурирати у 4 фазе:

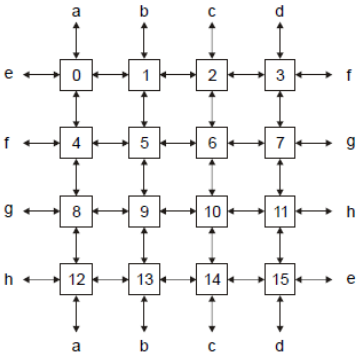
    - Партиционисање – има за циљ да цијели процес изђели на веци број задатака између којих постоји потенцијално велика могуцност конкурентног процесирања. Задаци обухватају како рацунања тако и податке над којима се рацунања изводе. Добра пођела имплицира равномјерну распођелу, како рацунања тако и података по задацима.

- Установљавање веза – фаза у којој се установљава зависност између задатака и потребни трансфери података између њих. За спецификацију комуникације је потребно дефинисати податке и комункационе везе између задатака.
- Укрупњавање – врси се у циљу смањења комуникационих активности уз задрзавање сто је могуце вецег износа паралелизма. Додатно, потребно је одрзати прилагодљивост рјесења обиму проблема и прихватљиву цијену програмске имплементације.
- Мапирање – задатака по процесним елементима са циљем минимизације функције циља је комплексан проблем. Ако је функција циља минимизација времена система задатка на паралелној архитектури, онда се говори о алгоритму распоређивања. За постизање наведене функције циљаалгоритми распоређивања се базирају на стратегији распоређивања конкретних задатака на разлицте процесоре и на стратегији обједињавања изврсења задатака на једном процесору у циљу смањења комуникационих активности.

Уобичајена функција циља за извођење неке апликације на паралелном систему у теоретским разматрањима је минимизација времена извршења, а за скуп независних програма, оптималан баланс оптередења по процесним елементима. У пракси, најчешће је од интереса добијање најповољнијег односа перформанса/цијена, гђе цијена укључује не само цијену процесирања него и развоја апликације. Претходно наведено питање имплицира инхерентно постојање паралелизма у апликативном проблему, те оптималну експлоатацију истог у паралелном систему. Апликативни проблем се може ријешити одређеним уређеним скупом операција – алгоритмом, који се може посматрати на различитим нивоима апстракције: нивоу задатака, подзадатака, процедура, група инструкција, самих инструкција или елементарних операција.

16. Функције повезивања за ову мрежу представљене су на слици и дефинисане су са :

$$FNN+1(P) = (P+1) \bmod N$$
$$FNN-1(P) = (P-1) \bmod N$$
$$FNN+n(P) = (P+n) \bmod N$$
$$FNN-n(P) = (P-n) \bmod N$$



гдје се $n = N$ подразумјева као цели број.Слично као код PM2I мрежа, рутирање у FNN мрежи се може остварити на бази разлике адреса одредишног и изворишног процесора.

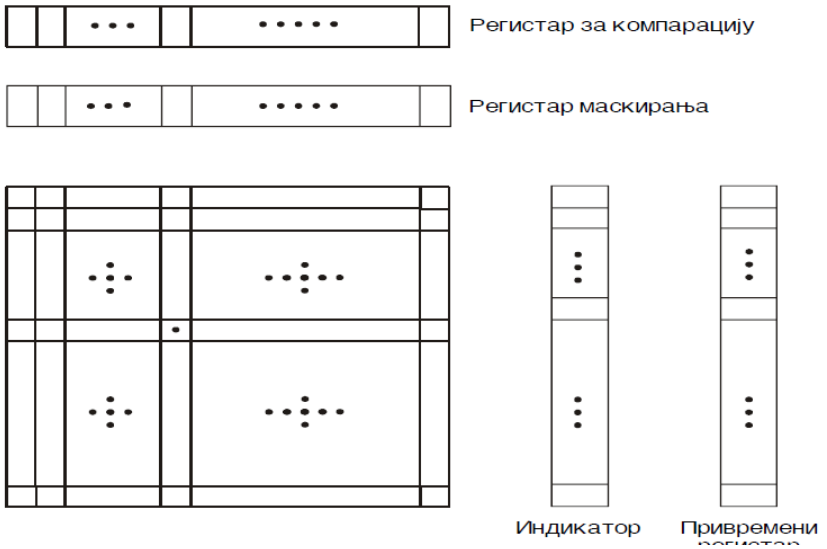
Ако посматрамо PM2I и FNN мрежу гдје обе повезује N процесора и где је $N = 2^m$ за PM2I и $n = N$ (цели број) за FNN мрежу (односно $n = 2^m / 2$), онда је PM2I+0 = FNN+1, PM2I–0 = FNN–1,

PM2I+m/2 = FNN+n, PM2I– m/2 = FNN–n.

Импликација је да су FNN функције повезивања подскуп PM2I функција. Овај тип спрежних мрежа је често кориштен у актуелним системима: Intel Paragon, CM-2, ASCI Red, Cray T3E (3D-torus) itd.

17. За разлику од класичних рачунара, ге се референцирање неког податка у меморији остварује на основу адресе, асоцијативни процесори референцирају податке у меморији на основу њиховог садржаја. На тај начин је могуће референцирање више података

исторвремено. Меморија која има наведене могућности ,назива се асоцијативна меморија.Типична организација асоцијативне меморије дата је на слици:



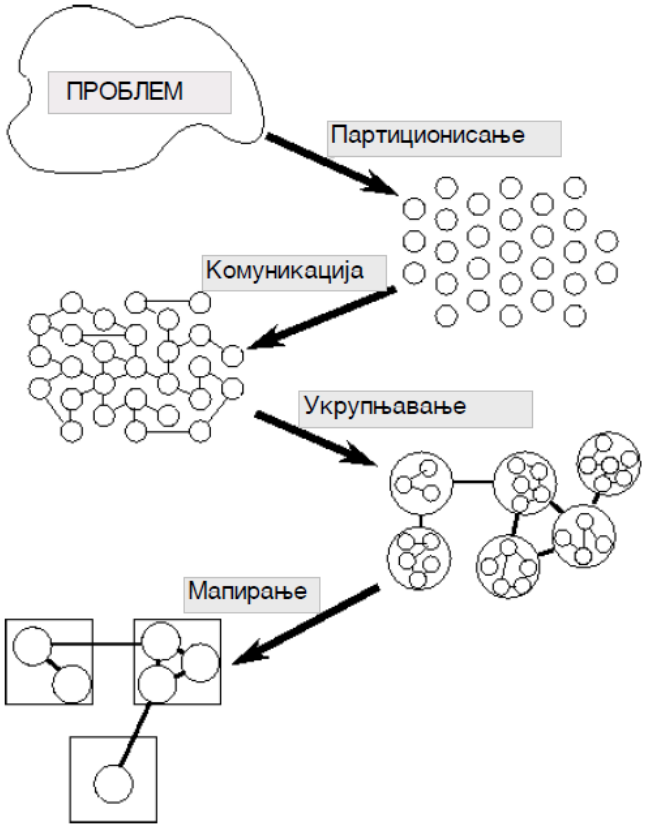
Управљачка логика специфекује преко регистра за компарацију садржаја свих немаскираних дјелова ЦР са свим локацијама асоцијативне меморије. Компарацију изводи контролно логика придружена меморијској локацији. Компарација се може изводити серијски,бит по бит или паралелно.

Асоцијативни процесори се повезују на рачунар домаћин као специјални процесори за одређене апликације. Модул за повезивање са околином обезбеђује широк спектар могућности : директан приступ меморији, паралелни У/И ,спрегу са другим специјализованим процесорима, кориштење сензорских улаза. Све ово омогућава постизање високих перформанси у апликацијама као што су матичне операције,управљачки системи у стварном времену и други.Због скупе реализације компараторске логике асоцијативне меморије су малог капацитета и скупе, што лимитира њихову употребу за специјализоване намјене. Значајна примјену имамо у реализацији кеш меморија.

18. Теоретски, максимално убрзање које се може постићи извођењем програма на паралелном систему са Н процесора једнако је броју процесора. У таквом сценарију, свих Н процесора би изводило само корисне инструкције, и сви би почели и заврши-ли процесирање у истим временским тренуцима. У пракси, остваривање оваквих максималних перформанси се не постиже због губитака времена проузрокованих:
    - комуникационим кашњењима,
    - синхронизацијом између процесора,
    - незапосленошћу процесора (празни ходови процесора),
    - неопходним управљачким активностима оперативног система.

    Реализација наведеног циља се може остварити структурирање пројектовања у четири фазе: партиционисање, установљавање веза (комуникација), укрупњавање и мапирање.

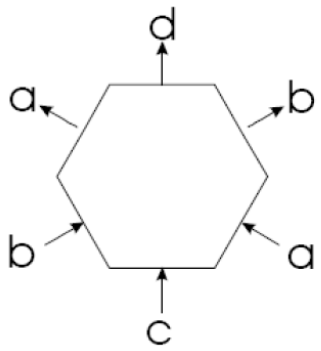
- Партиционисање има за циљ да се цијели процес издијели на што већи број задатака између којих постоји потенцијално велика могућност конкурентног процесирања. Задаци обухватају како рачунања (скуп операција) тако и податке над којима се рачунања изводе. Добра подјела имплицира равномјерну расподјелу, како рачунања, тако и података по задацима. Основне комплементарне технике раздиобе су декомпозиција домена (података) и функционална декомпозиција. Генерално, задаци не треба да садрже редундантна рачунања или податке и, број задатака који се добијају декомпозицијом треба да је у пропорцији са величином проблема.



- Повезивање је фаза у којој се установљава зависност између задатака и потребни трансфери (комуникације) података између њих. За спецификацију комуникације потребно је дефинисати, како саме податке који се измјењују одређеним комуникационим механизмом (порукама нпр.), тако и комуникационе везе (канале) између задатака.
- Укрупњавање груписање задатака се врши у циљу смањења комуникационих активности (и временских губитака) уз задржавање што је могуће већег износа паралелизма. Додатно, потребно је одржати прилагодљивост рјешења обиму проблема, и прихватљиву цијену програмске имплементације.
- Мапирање задатака по процесним елементима са циqем минимизације функције циља је, у општем случају, комплексан проблем. Ако је функција циља мининизација времена извођења система задатака на паралелној архитектури, онда се говори о алгоритмима распоређивања.

19. Многи проблеми који захтијевају интензивна рачунања (процесне ресурсе) и који садрже висок степен паралелизма могу се ефикасно ријешити примјеном једноставних процесних елемената организованих у неку регуларну структуру (мрежа,стабло, линеарно поље и сл.). Оваква организација процесирања је погодна за имплементацију у ВЛСИ технологији, с обзиром на једноставност елемената и правилност структуре.Информације у овај (систолични) систем се "упумпавају" преко вањских ћелија,преко којих се остварује и

излазна веза. Базични елемент најчешће омогућава само једноставне операције, као нпр. сабирање и множење. Процесни елемент може да процесира податке и остварује везу са другим елементима паралелно (више бита одједном) или серијски, може бити са или без локалне меморије. Подаци улазе у систолично поље, обрађују се у процесним елементима по принципу проточне обраде и преносе у слиједеће процесне елементе ритмично и синхроно. Комуникација између процесних елемената је једноставна и локализована на најближе сусједе, што омогућава (због малог растојања) повећање фреквецније радног такта.Систолични процесори захтијевају специфично пројектовање алгоритама за одређене рачунски интензивне проблеме и одговарају}у организацију систоличног поља. Високе перформансе и ниска цијена имплементације утицали су на примјену систоличких поља у многим областима (дигитална обрада сигнала, матрична израчунавања и сл.). Технолошки проблеми у имплементацији великог броја У/И веза на ВЛСИ колу, лимитирају величину и могућност систоличних процесора, а тиме и област њихове примјене.



$$d = c + a * b$$

20. Бафер за промену редоследа (РОБ) се корсити у Томасуло алгоритму за ванредно извршавање инструкција. Он дозвољава инструкцијама да буду привржене у реду.

Обично постоје три фазе инструкција: издавање, извршавање и писање резултата. У Томасуло алгоритму, постоји додатна фаза "привржење". У овој фази, резултати инструкција ће бити сачувани у регистру или меморији. У фази писања резултата, резултати се ставе у бафер за промену редоследа. Цео садржај овог бафера се онда може користити при извршавању других инструкција које зависе од ове.

Постоје додатна поља у сваком уносу у бафер:

- Тип инструкције (скок, постављање у меморију, постављање у регистар)
- Дестинација (адреса меморије или регистарски број)
- Резултат (вредност која иде у дестинацију или индикација (не)успешног скока)
- Валидност (да ли резултат већ постоји)

Додатне предности овог бафера су омогућавање прецизних изузетака и једноставне rollback контроле циљних адресних предвиђања (гранање или скок). РОБ ради тако што чува инструкције редом којим их је увезао. РОБ-у такође може да се приступи са стране јер свака резервациона станица има додатни параметар који показује на инструкцију у РОБ-у. Када

предвиђање скока није тачно, или се деси непоправиви изузетак у инструкцијском току, РОБ се празни тако да не садржи инструкције и резервационе станице су постављене на почетак.

Постоје два типа спекулације:

- управљачка и
- спекулација о подацима.

У обе варијанте компајлер повећава ILP (Insturction Level Paralism - Паралелизам на нивоу инструкција) спецификацијом извршења неке операције раније, са циљем елиминисања кашњена на критичном путу извођења инструкција. Преводилац спецификује спекулативно извршење неке инструкције раније, ако постоји значајна вјероватноћа да ће спекулација бити остварена (да је извођење инструкције у сагласности са секвенцијалним током извршења) и ако ранијим извршењем инструкције долази до повећања могућности експлоатације паралелизма. Ако посматрамо инструкцију:

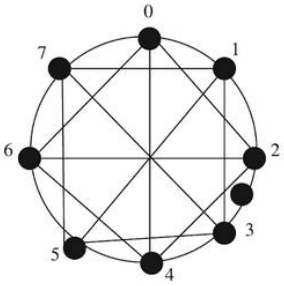
Извођење операције ld_addr1, target1 прије него што је познат услов a > b се може карактеризовати као спекулација да ће услов a> b бити истинит. Раније извођење инструкције ld_addr1, target1 омогућава преклапање извођења те инструкције (која траје дуже због референцирања меморије) са другим инструкцијама, тако да су операције специфициране том инструкцијом извршене у тренутку тестирања услова. Спекулација о подацима је добављање података из меморије прије операције смјештања у меморију која претходи том прибављању, ако постоји потенцијална могућност да лоад и сторе операције референцирају исти дио меморије.

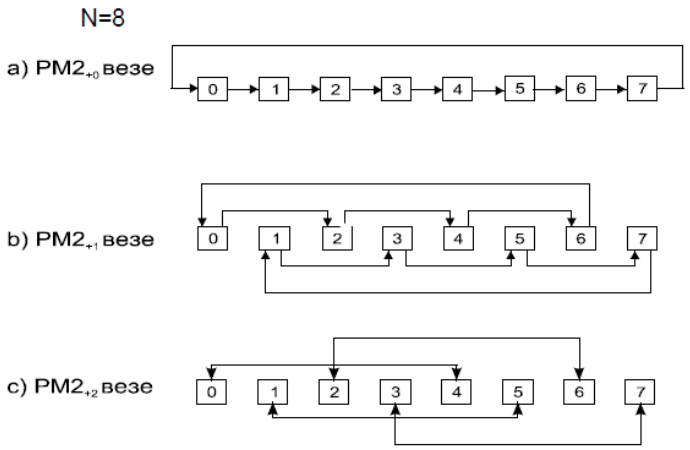
21. Функције повезивања за ову мрежу су дефинисане изразом:

$$\mathrm{PM2I}_{+i}\ (j) = (j + 2^i)\ (\mathrm{mod}\ N)$$
$$\mathrm{PM2I}_{-i}\ (j) = (j - 2^i)\ (\mathrm{mod}\ N)$$

где је $0 \leq j \leq N - 1$, $0 \leq i \leq log_2 N - 1$.

и представљене су на слици. Укупно има 2м функција повезивања. У овој шеми повезивања, процесор са адресом П може послати податке свим другим процесорима чије су адресе (j ± 2^i) (mod N). Одређивање руте за комуникацију између процесора може се базирати на разлици адреса одредишног и изворишног процесора. Нпр. За

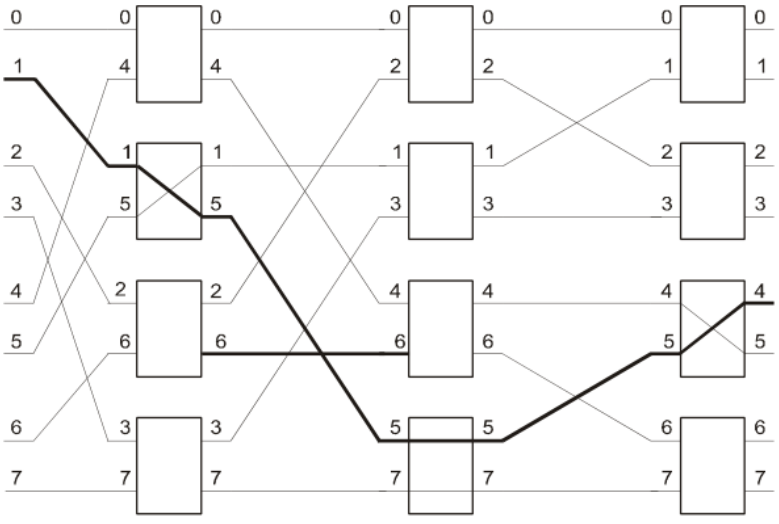




Сл. 6.20 Функције повезивања PM2I спрежне мреже.

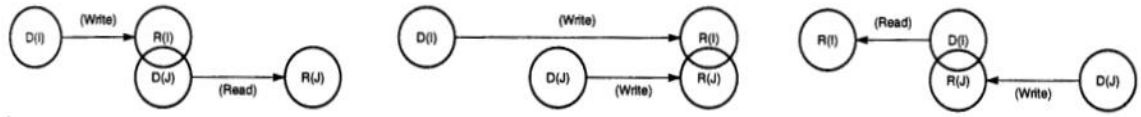
комуникацију између процесора 1 и 12 (N = 16), разлика адреса је 11, па податак може стићи на одредиште коришћењем функција $PM2I_{+0}$, $PM2I_{+1}$, $PM2I_{+3}$, пролазећи кроз чворове 2,4. Различит скуп функција резултоваће истим одредиштем, ако је вриједност $\sum 2^i$ једнака разлици адреса одредишта и извориште, гђе је и индекс PM2I функција из скупа. Дата манипулатор вишестепене спрежне мреже су базиране на PM2I мрежама.

22. Ова мрежа се састоји из m = log2N степени, гђе је N број улазних односно излаз-них елемената који се повезују. У сваком степену постоји H/2 прекидачких елемената (укупно m * N/2) и по N улазних и излазних веза, означених од 0 … N-1. Ако се везе означе тако да горњи (доњи) улази (излази) прекидачких елемената имају исту ознаку, онда степен и генерализоване коцке имплементира функцију повезивања $Cube_i$, јер врши повезивање линија чије се адресе разликују у i-тој бит позицији.Ако се остварује веза свих улаза на по један излаз, онда се користе само директна и унакрсна стања прекидачких елемената. Ако је неки прекидачки елемент у степену и постављен у стање унакрсног повезивања, онда он извршава $Cube_i$ функцију повезивања. Ако се врши повезивање улаза S =(Sm-1, Sm-2 …S1S0) са излазом D= ($D_{m-1}D_{m-2}$ …D_1 D_0), онда стање одговарајућег прекидачког елемента у степену и мора бити постављено за унакрсно повезивање, ако је s_i xor d_i =1, у супротном - за директно повезивање. Прекидачки елементи су постављени у стање за



унакрсно повезивање у степенима 2 и 0 (Cube2 ,Cube0) и директно у степену 1. За случај прослеђивања од једног извора ка више одредишта истовремено, користи се стање емисије одозго (одоздо) прекидачких елемената. За прослеђивање порука кроз мрежу, може се користити адреса рутирања (routing tag) T = S xor D, иницијалне дужине m-бита, гђе су S и D адресе изворишта/одредишта. У наредном степену, односни бит кориштен у претходном степену може се одбацити. Ако се користи пренос на више одредишта (емисија), потребно је 2m бита за адресу рутирања.

23. Предвиђање гранања је изузетно важно због смањивања цијене гранања.Генерално постоје двије врсте предвиђања гранања : статичко и динамичко.
    Статичко се врши у фази превођења. Статички подаци говоре да се код инструкција условног гранања гранање чешће догађа, па се бољи резултати добијају ако се гранање предвиди.Тачност ове технике је негде око 75%.
    Код динамичке стратегије предвиђања,предвиђање да ли ће доћи до гранања или не при извођењу инструкције гранања се базира на претходној историји гранања односне инструкције. За имплементацију се користи бафер циљног гранања БТБ.Ово техник се добија тачност између 86% и 96%.
24. Приступ заједничким варијаблама за читање/писање од стране различитих инструкција које се налазе у проточном систему, може довести до различитих резултата ако се инструкције изведу различитим редоследом. Овај тип зависности између инструкција се назива зависност по подацима. Мијењање оригиналног редослиједа инструкција је допустиво само ако то мијењање нема утицаја на коначан резултат извршења програма. Уколико се инструкције, између којих постоји зависност по подацима, не изводе по оригиналном редоследу, долази до логичких хазарда:



(a) Read-after-Write (RAW) hazard (b) Write-after-Write (WAW) hazard (c) Write-after-Read (WAR) hazard

25. Функција управљачке једнице процесора је да генерише сигнале за управљање свим активностима унутар ЦПЈ, контролом управљачке јединице се врши дохватање инструкције из меморије,њено декодовање и идентификација и пренос операнада од изворишта до функционалних једница у којима се обавља дата операција, те враћање резултата на одредиште. Овај процес се под контролом управљачке јединице понавља при извођену сваке следеће инструкције.Управљачка једница овде дјелује као неки командни центар из којег се управља радом осталих јединица система,а све у сврху извршавања машинских инстуркција.Да би управљачка јединица остварила све наведене задатке,она мора да:

- обезбједи коректан редосљед извршења инструкција
- генерисањем одговарајућих сигнала обезбједи извршавање селектоване функције

Што се тиче врста управљачких једница генерално их можемо класификовати у две врсте :

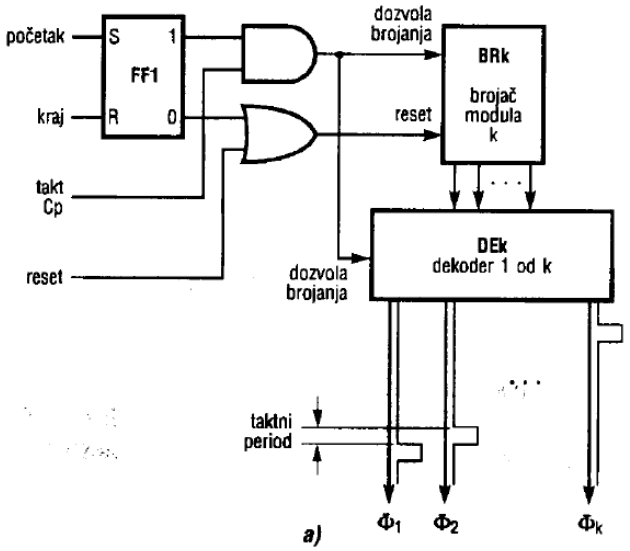
❖ управљачке јединице са директним управљањем – логичком мрежом се генеришу контролни сигнали у предефинисаном редосљеду за сваку инструкцију. Даље ове управљачке једнице се дјеле на основу реализације на :

- на оне засноване на табелама стања
- засноване на елементима за кашњење
- засноване на бројачкој секвенци

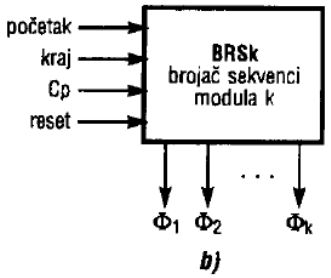
❖ управљачке једнице са микропрограмским управљањем – контролни сигнали се генеришу извршењем микроинструкција записаних у интерној микропограмској меморији процесора,а извршење сваке машинске инструкције се реализује низом елементарних акција које се имплементирају микроинструкцијама.Микроинструкције могу да буду хоризонталне – њима се спецификује извршавање више микрооперација и вертикалне – ако се њима спецификује извршавање једне микрооперације.

Закључујемо да је предност директног управљања брзина, а микропрограмског једноставност реализације и измјена/отклањања грешака у пројектовању.

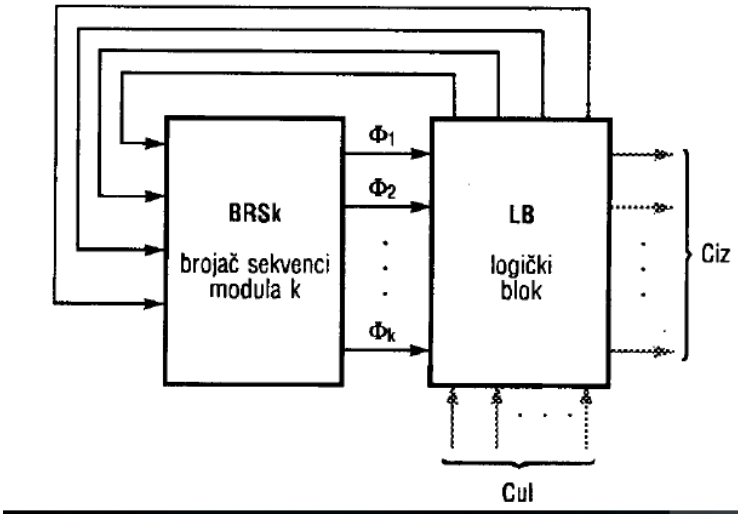
Управљачка једница реализована преко бројачке секвенце – управљачке једнице овог типа претпоставља извршење инструкције у к корака.У и-том кораку се активира неки скуп управљачких линија коришћењем односног фазног импулса $\Phi_и$.Фазни импулси генеришу се колом бројач секвенци .Сукцесивни импулси на излазу бројача су помјерени за време трајања импулса.



Спрега бројача секвенци и логичког блока којим се имплемнтира односна функција је предестављена на слици:



Sl.3.9 Brojač sekvenci modula *k*



Upravljačka jednica са директним управљањем заснована на табелама стања:

Приликом извођења инструкције процесор пролази кроз низ интерних стања. На основу текућег стања и скупа улазних сигнала, генеришу се излазни сигнали и процесор прелази у нову стање. Дио генерисаних излаза се користи као улазни сигнали следећег стања. Логика рада се може представити табелом стања, којом се описују прелази процесори кроз интерна стања.Врсте у табели одговарају стањима из скупа S_{int}.Колоне у табели одговарају подскуповима улазних сигнала b_j подскуп од Аулазно.Из стања Си под дејством подксупа улазних сигнала Бј прелази се у ново стање Сиј, при чему се активирају улазни сигнали дефинисани са аиј подску Аизлазно.Проблем је величина табеле у случају великог броја

	Upravljački signali			
Tekuće	Aul			
Stanje	b_1	b_2		b_m
S1	$S_{1.1}, a_{1.1}$	$S_{1.2}, a_{1.2}$	...	$S_{1.m}, a_{1.m}$
S2	$S_{2.1}, a_{2.1}$	$S_{2.2}, a_{2.2}$	...	$S_{2.m}, a_{2.m}$
---	---	---	---	---
Sn	$S_{n.1}, a_{n.1}$	$S_{n.2}, a_{n.2}$	...	$S_{n.m}, a_{n.m}$
naredno stanje				

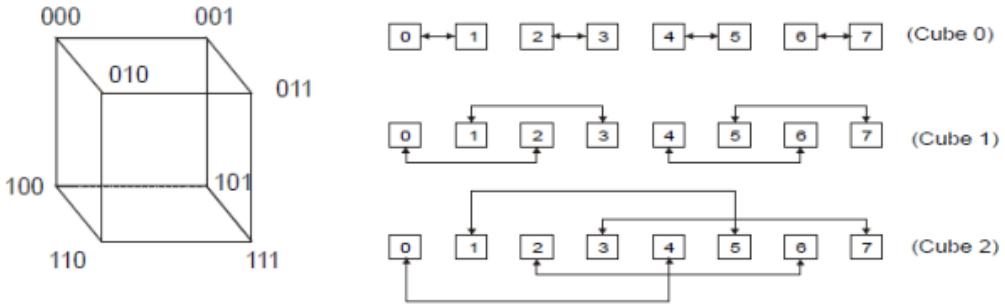
стања.

Остаје нам још да појаснимо уј засновану на елемнтима за кашњење.Код овог концепта полази се од дијаграма тока рада управљачке једнице . За репрезентацију н стања система користи се н-бистабилних елемената један елемент по стању.У датом тренутку активан је само један елемент и тај указује на текуће стање.Прелаз у ново стање се може представити подскупом основних концепата дијаграма тока ,при чему се сваки од дијаграма имплементира одговарајућим управљачким колом.

26. Нека је укупан број процесора $N = 2^m$ и нека је бинарна репрезентација адресе сваког од N процесора облика $p_{m-1} ... p_{i+1} p_i p_{i-1} ... p_0$. Функције за повезивање за ову спрежну мрежу дате су изразом:

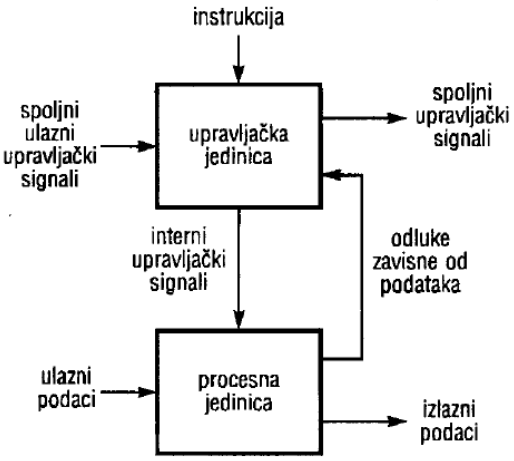
$Cube_i (p_{m-1} ... p_{i+1} p_i p_{i-1} ... p_0) = p_{m-1} ... p_{i+1} \overline{p_i} p_{i-1} ... p_0$ за $0 \leq i \leq m - 1$

Сваки процесни елемент се може функцијама повезивања повезати са m других процесора, так о да је укупан број могућих повезивања $m*2^m$, број веза $m*2^m /2$. Функције повезивања и 3D структурна репрезентација за N = 8, дате су на слици:



$Cube_i (p_{m-1} ... p_{i+1} p_i p_{i-1} ... p_0)$ мапира било коју адресу на другу која има на i-том мјесту комплементаран бит. У представи на коцки, $Cube_0$ повезује чворове на хоризонталним, $Cube_1$ на дијагоналним а $Cube_2$ на вертикалним линијама. За трансфер су потребна 1, 2 или 3 корака завусно од извора и одредишта.

27.



28. Код архитектура овог типа постоји један секвенцијални ток инструкција и једна контролна јединица која управља истовременим извршењем сваке инструкције на више процесних елемената. Сви омогућени процесни елементи изводе (исту) инструкцију над различитим подацима, чиме се реализује паралелна обрада на одређеном скупу података. Једним инструкционим током се избјегава понављање процеса добављања инструкције за сваку процесну јединицу. С обзиром на то да сви процесори раде синхронизовано, на основу управqачких сигнала из контролне јединице, овакве архитектуре су погодне за експлоатацију ситнозрнастог паралелизма.

    Процесори са СИМД архитектуром:

    - Intel SSE and MMX (Pentium III, SSE4 - Core series),
    - ARM NEON (Cortex-A8, Cortex-A9),
    - MIPS MDMX.