СӘТБАЕВ УНИВЕРСИТЕТІ

SATBAYEV UNIVERSITY

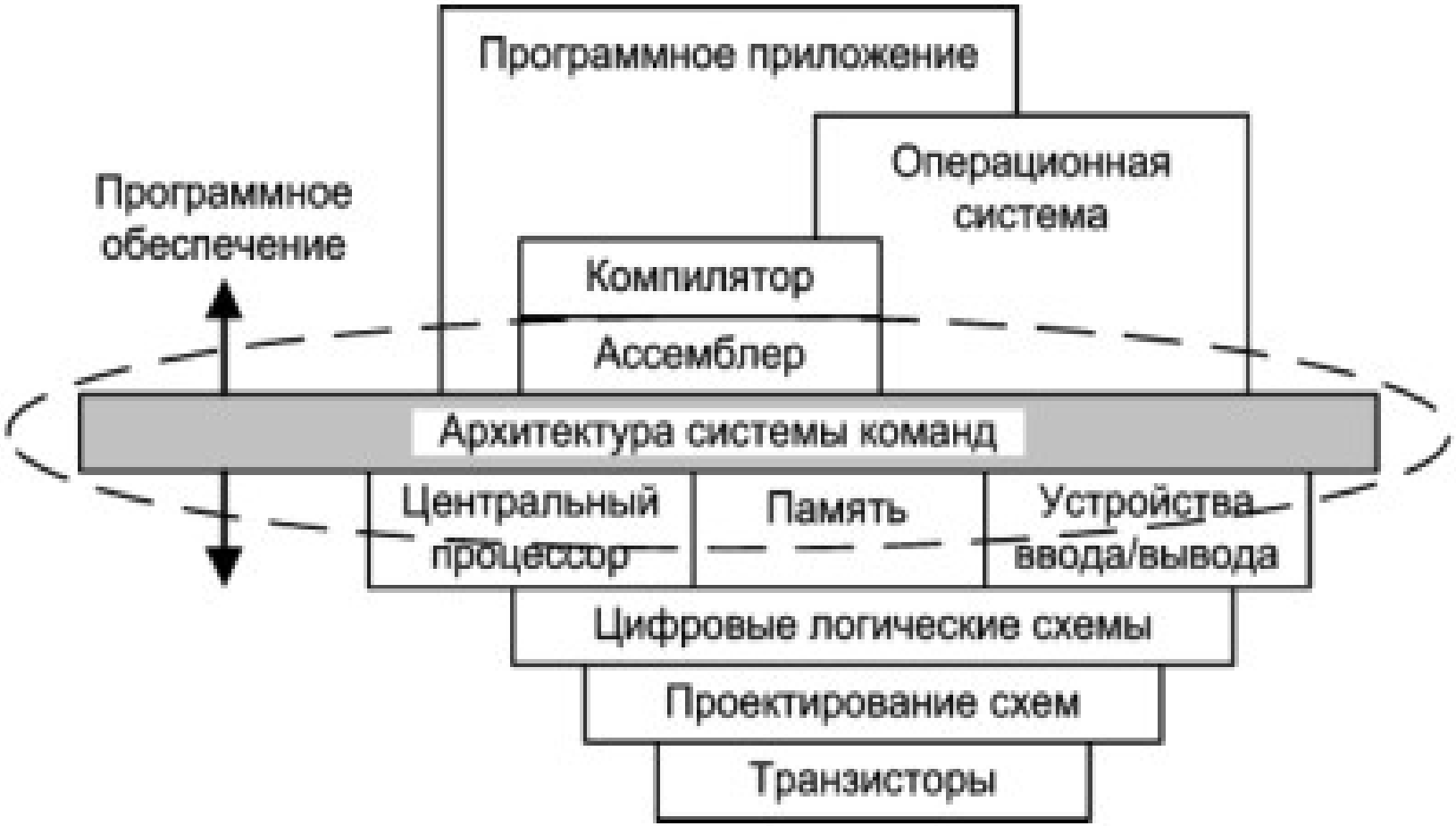
Командалар жүйесінің архитектурасы Командалар жүйесінің архитектурасының классификациясы

Оқытушы: Жекамбаева М.Н.
«Программалық инженерия»
кафедрасының профессоры
m.zhekambayeva@satbayev.university

Командалар жүйесінің архитектурасы

Командалар жүйесі деп есептеу машинасының толық командалар тізбегін айтамыз. Өз кезегінде командалар жүйесінің архитектурасы (КЖА) деп программистке көрінетін және қолжетімді есептеу машинасының құралдарын айтуға болады. КЖА программалық қамтамаларды өңдеуші керектіктерін аппараттық есептеу машинасы құрушылардың мүмкіндіктерімен сәйкестендіріп қарастыруға болады (келесі сурет)

Командалар жүйесінің архитектурасы



Программалық және аппараттық қаматамалар арасындағы интерфейс ретіндегі командалар жүйесінің архитектурасы

Командалар жүйесінің архитектурасы

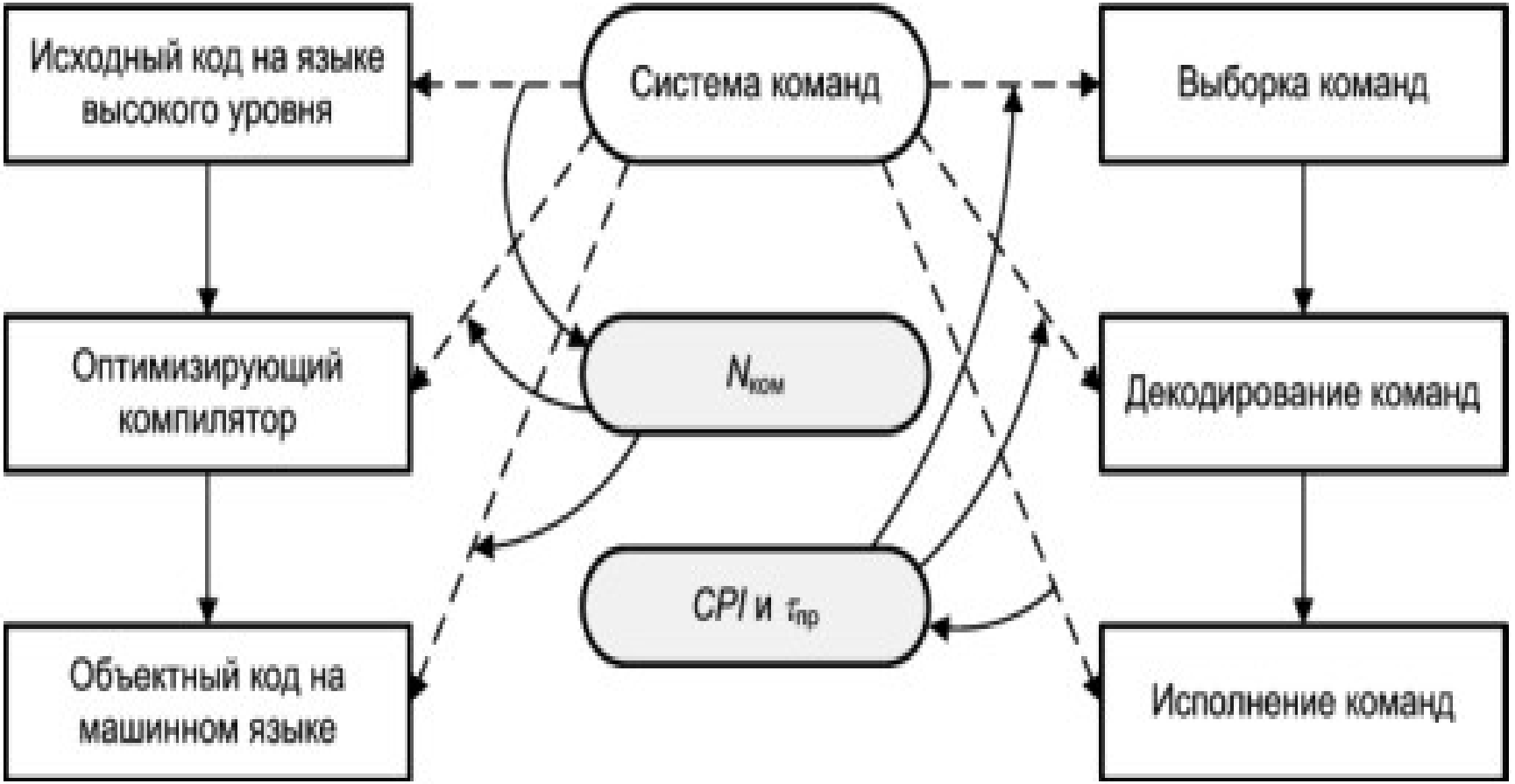
Нәтижесінде екі жақтын да мақсаты аз уақыт ішінде есептеулерді ең тиімді әдіспен тарату болып табылады, мұндағы басты мақсат командалар жүйесінің архитектурасын дұрыс таңдау болып табылады.

Жеңілдетілген трактовкадағы программаларды орындау уақытын (T выч) программадағы командалар саны арқылы анықтауға болады (N ком), бір командаға келетін процессор трактілерінің орташа саны (CPI), және тактілік периодтың ұзақтығы τ пр келесідегідей есептеледі:

$$T_{выч} = N_{ком} \times CPI \times \tau_{пр} .$$

Құрылған өрнектердің әрбірі командалар жүйесінің архитетурасының аспектілеріне тәуелді болады, ал ол өз кезегінде басқаларға әсер етеді (келесі сурет), ол КЖА таңдауға өте жауапты қарау керек екендігін ескертеді.

Командалар жүйесінің архитектурасы



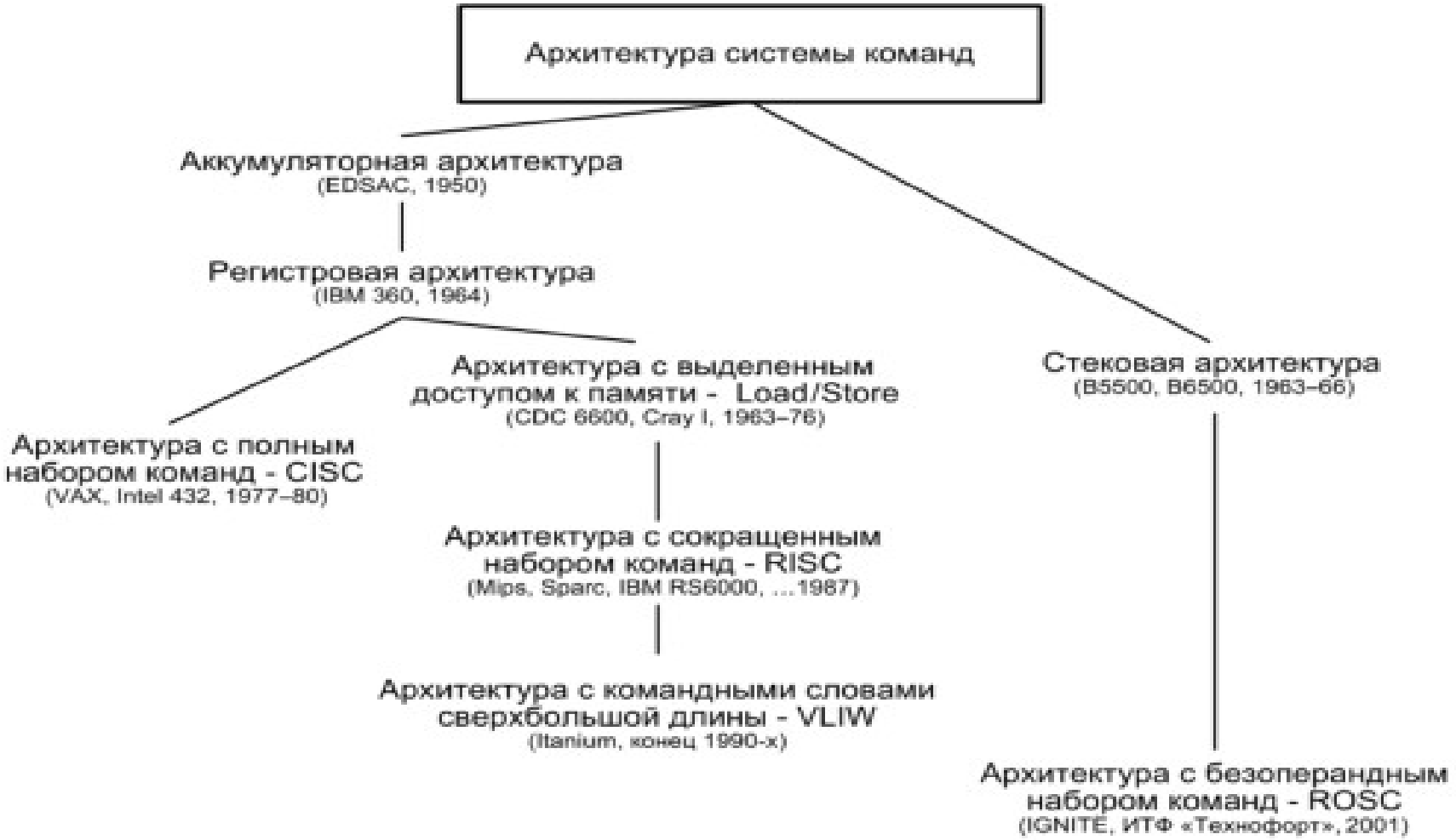
Есептеу тиімділігін анықтайтын командалар жүйесі және факторлар арасындағы әрекеттесулер

Командалар жүйесінің архитектурасының классификациясы

Есептеу техникасының даму тарихында өңдеушілер көзқарасы жағынан командалар жүйесінің архиектурасының қандай-да бір түрлерінің пайдаланылуы жағынан өзгерулер байқалуда. Қазіргі кезде пайда болған жағдайларға байланысты КЖА келесідегідей көрсетуге болады (келесі сурет).

Жаңа КЖА өту келесі мотивтерге байланысты анықталады, соның ішінде маңызды екеуін қарастырайық. Біріншісі — бұл есептеу машиналарымен орындалатын операциялар құрамы, және олардың қиындығы. Екіншісі — бұл операндтарды сақтау орны, ол деректерді өңдеу командаларының адрестік бөлігін, адрес ұзындығын және санын көрсетуге әсерін тигізеді. Міне осы сипаттамалар командалар жүйесінің архитектурасының көрсеткіштері ретінде алынған.

Командалар жүйесінің архитектурасының классификациясы



Командалар жүйесінің архитектурасының дамуының хронологиясы

Командалар қиындығы және құрамы бойынша классификациясы

Қазіргі заманғы программалау технологиялары жоғарғы деңгейдегі тілдерге (ЖДТ) бағытталған, олардың басты мақсаты — программалау процессін жеңілдету. Бірақ ЖДТ өту өз қиындықтарын туындатады: ЖДТ сипаттайтын қиын операторлардың болуы, ол қарапайым машиналық операциялардан ерекшеленеді. Сәйкесінше ЕМ программалардың орындалуы қиындайды. Міне осы мәселелерді шешу үшін өңдеушілер келесі үш КЖА түрін таңдау керектігі туындайды:

- толық командалар жиыны бар архитектура: CISC (Complex Instruction Set Computer);
- қысқартылған командалар жүйесі бар архитектура: RISC (Reduced Instruction Set Computer);
- өтеүлкен ұзындықтағы сөздері бар командалары архитектура: VLIW (Very Long Instruction Word).

Командалар қиындығы және құрамы бойынша классификациясы

CISC-архитектурасында семантикалық бөлінулер командалар жиынының көп болуымен шешіледі, оған IBM компаниясымен шығарылған әмбебап ЕМ (мэйнфрейдер) және Intel компаниясымен шығарылған x86 сериясындағы МП жатады. CISC-архитектуралар үшін:

- процессорда жалпы міндетті регистрлардың санының салыстырмалы аз болуы;
- машиналық командалардың көп болуы, олардың көп бөлігі аппаратты ЖДТ қиын операторларын тарата алатын болуы;
- операдтарды адресациялау әдістерінің әртүрлілігі;
- әртүрлі разрядтылықтағы командалар форматының көп болуы керек;
- жадыға қатынаумен біріктірілетін өңдеулер кездесетін командалардың болуы керек.

Командалар қиындығы және құрамы бойынша классификациясы

Бұл архитектураның басты идеясы ЕМ командалар жиынын қарапайым командалармен ауыстырып шектеу, және тек процессор регистрларын ғана пайдалану. Бұл шаралар жылдамдықты арттыруға және аппараттық құраралдарды жеңілдетуге себін тигізді. RISC-архитектурасының элементтері бірінші Cray Research компаниясының CDC 6600 және суперЭВМ пайда болды, сондай-ақ Intel және AMD компаниясының микропроцессорларында да кеңінен қолданылуда, сондықтан CISC және RISC арасындағы айырмашылықтар біртіндеп жойылып келеді.

Командалар қиындығы және құрамы бойынша классификациясы

CISC- Және RISC-архитектураларынан басқа — өтеүлкен ұзындықтағы сөздері бар командалары архитектурасы (VLIW) бар.

VLIW концепциясы RISC-архитектурасының базасына негізделген, бірақ мұнда бірнеше қарапайым RISC-командалар бір өтеүлкен ұзындықтағы сөздері бар командалары архитектурасына бірігеді және параллельді орындалады. КЖА жүйесінде VLIW архитектурасы RISC қатты ерекшеленбейді.

Осы үш негізгі архитектуралардың салыстырмалы бағалану кесте 7.1. көрсетілген

CISC-, RISC- және VLIW-архитектураларының салыстырмалы бағалануы

Сипаттамасы	CISC	RISC	VLIW
Командалар ұзындығы	Өзгереді	Бірегей	Бірегей
Командадағы өрістерінің орналасуы	Өзгереді	Өзгермейді	Өзгермейді
Регистрлерінің саны	Бірнеше (жиі манадандырылған)	Жалпы міндетті регистрлері көп	Жалпы міндетті регистрлері көп
Жадыға қатынауы	Әртүрлі типтегі командалардың бөлігі ретінде орындалуы мүмкін	Тек арнайы командалармен орындалады	Тек арнайы командалармен орындалады

Операндтарды сақтау орны бойынша классификациясы

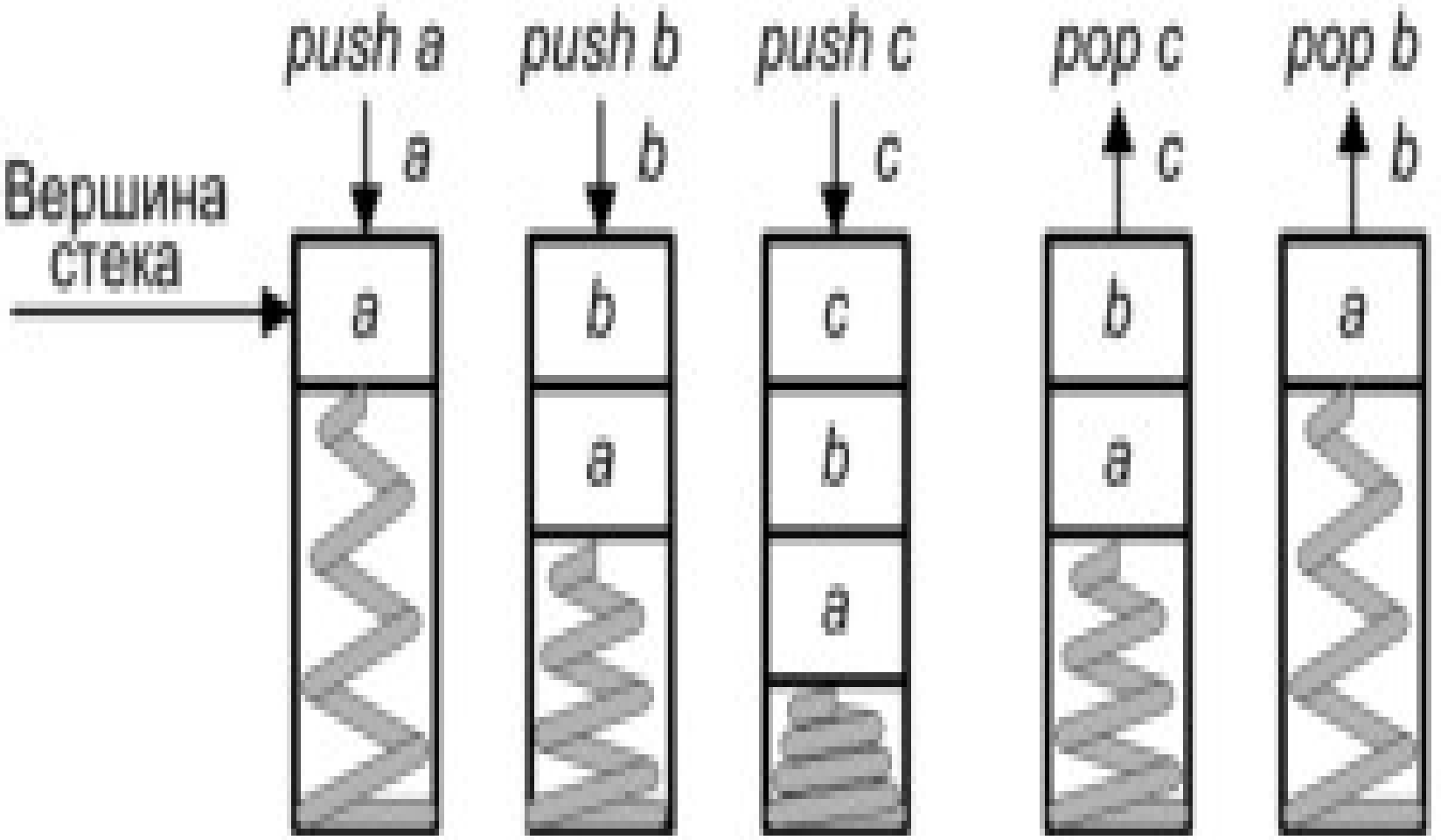
Командалар жиыны және олардың қиындығы маңызды фокторлардың бірі болып табылады, бірақ КЖА таңдауда операндтардың қайда сақталатындығына және оларға қалай қатынау керек екендігіне де көп көңіл бөлу керек. Бұл позициядан командалар жүйесінің архитектурасын келесі түрлерге бөлуге болады:

- ⟨ стектік;
- ⟨ аккумуляторлық;
- ⟨ регистрлік;
- ⟨ жадыға бөлінген қатынау.

Стектік архитектура

Стек деп ЕМ негізгі жадысынан құрылымдық ұйымдастырылуы жағынан ерекшеленетін жадыны айтамыз.
Стек көптеген бір-бірімен логикалық байланысқан ұяшықтарды құрады (сурет 7.4), ол бір-бірімен «соңғы кірдің, бірінші шығасың» (LIFO, Last In First Out) әдісі бойынша байланысады.

Стектік жадының әрекеттесу принципі



Стектік жадының әрекеттесу принципі

Жоғарғы ұяшықты стек төбесі деп атайды. Стекпен жұмыс жасау үшін екі операция қарастырылған: push (стекке деректерді орналастыру) және pop (стектен деректерді шығару).

Стектік машина әрекетінің принциптерін келесі өрнек мысалымен қарастырайық (a + b) * (c + d) – e.

Берілген жазу формасы операндтарды жүктемелеу тәртібі мен стекке орналастыру операциясын көрсетеді (сурет 7.5).

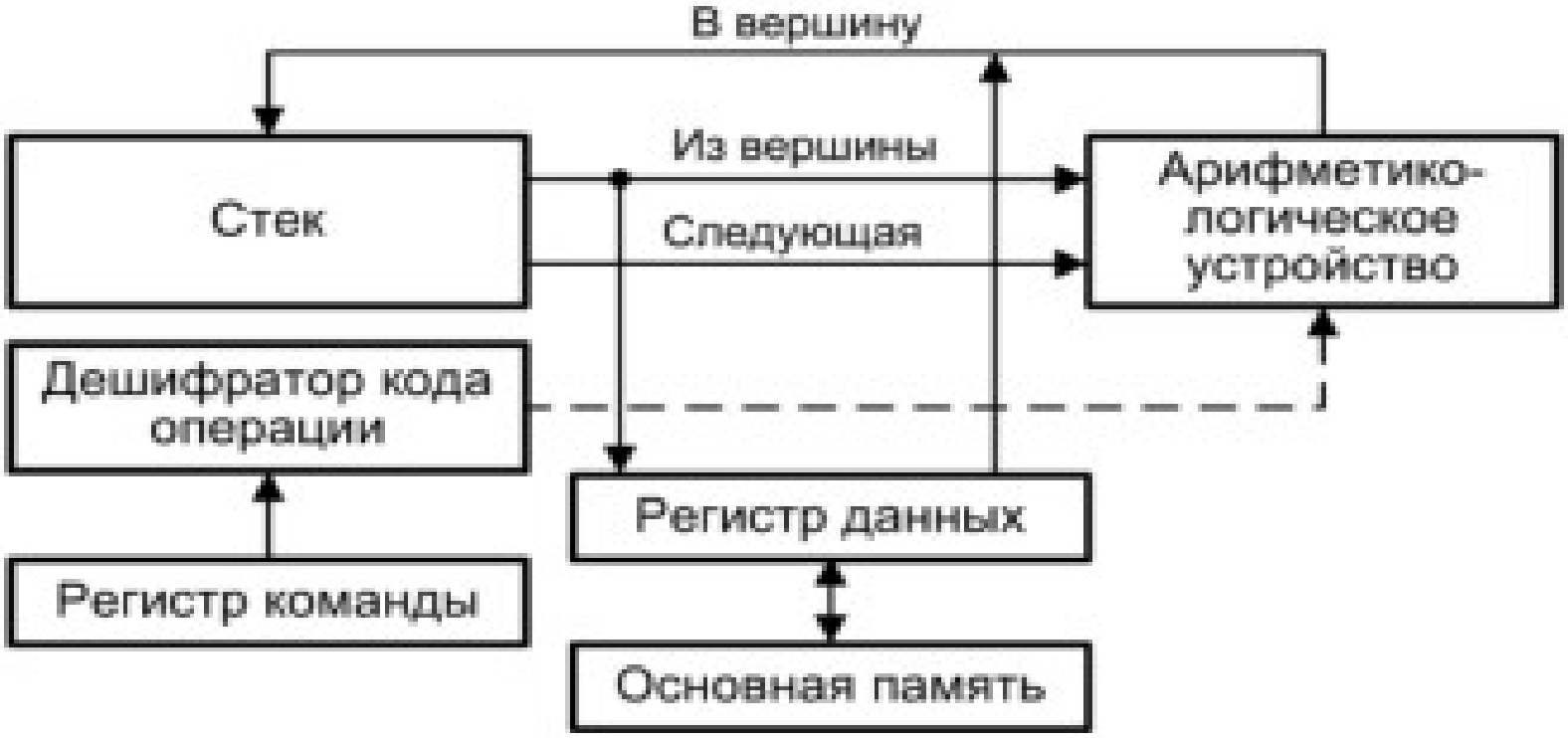
Стектік жадының әрекеттесу принципі

push a	*push b*	*add*	*push c*	*push d*	*add*	*mul*	*push e*	*sub*
a	*b*	*a+b*	*c*	*d*	*c+d*	*(a+b)*(c+d)*	*e*	*(a+b)*(c+d)-e*
	a		*a+b*	*c*	*a+b*		*(a+b)*(c+d)*	
				a+b				

Есептеу машинасындағы стектік архиектура үшін өрнектің орынталу тізбегі

Стек негізіндегі есептеу машинасының архитектурасы

КЖА стектік негіздегі ЕМ мүмкін вариантарының ақпараттық тракті және негізгі түйіндері сурет 7.6 көрсетілген.



Стек негізіндегі есептеу машинасының архитектурасы

Ақпараттар стек төбесіне АЛҚ немесе негізгі жадыдан алынып жазылады. Стекке жазу үшін push x қолданылады, ол жады ұяшығынан оқиды да деректер регистріне орналастырады. АЛҚ нәтижесі автоматты түрде стек төбесіне жазылып отырады.

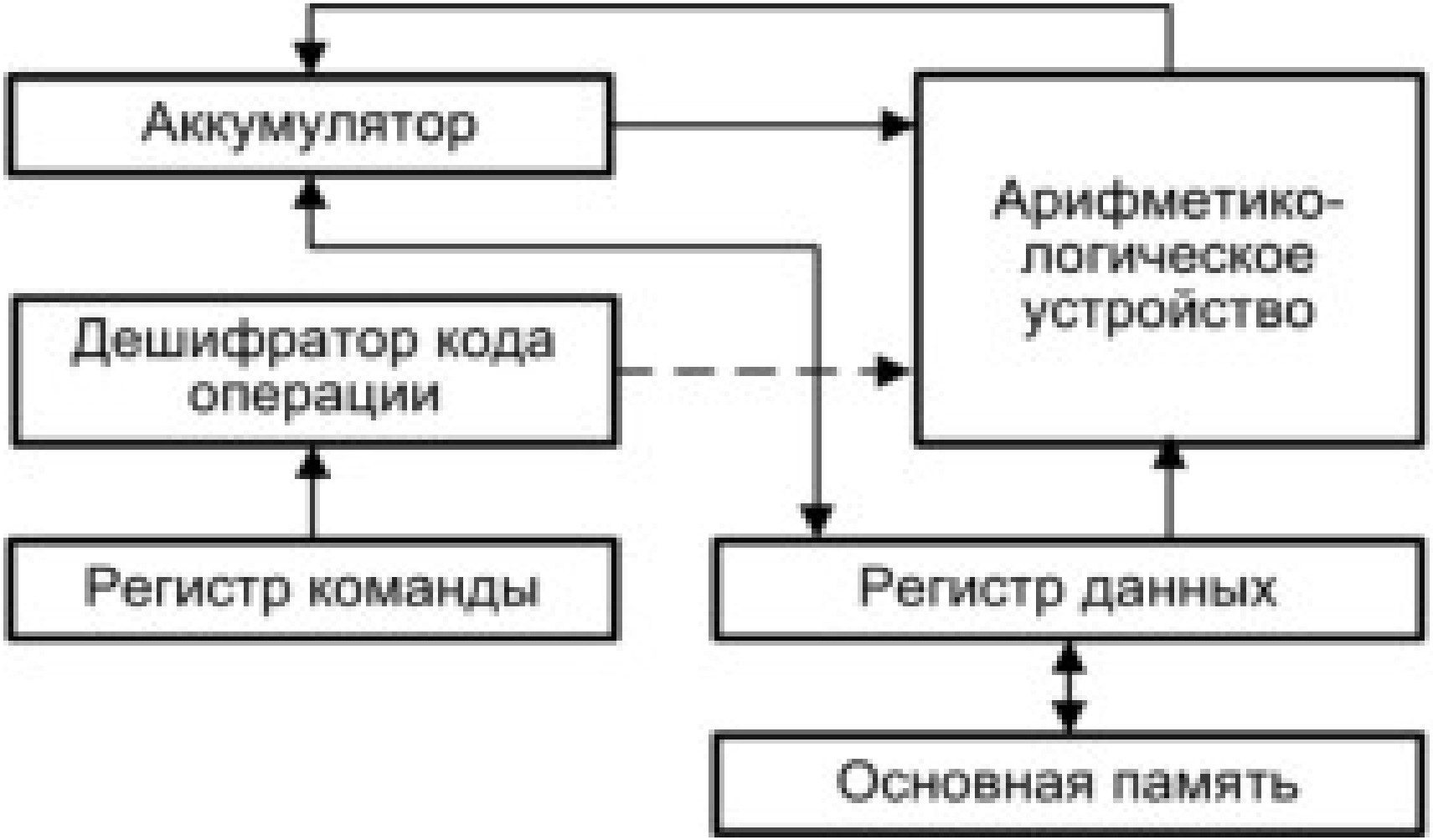
Стек төбесіндегі деректерді жады ұяшығына сақтау үшін pop x командасы қолданылады. Бұл команда арқылы жоғарғы ұяшық мәні шинаға беріледі, ол арқылы ұяшыққа жазу орындалады, одан кейін стектің барлық құрамы жоғарыға бір позицияға жылжиды.

АЛҚ операцияларды орындау үшін АЛҚ кірісіне стектің екі төбесіндегі ұяшығының мәндері алынады. Нәтиже стектің төбесіне жазылады.

Аккумуляторлық архитектура

Аккумулятор базасындағы архитектура тарихи біріншілердің бірі ретінде пайда болды. Мұнда арифметикалық немесе логикалық операцияларды орындау кезіндегі операдтардың бірін сақтау үшін ерекшеленген регистр — аккумулятор қолданылады. Осы регистрге нәтиже де жазылады. Бастапқыда екі операнд та негізгі жадыда болады, және операцияны орындаудан бұрын олардың бірі аккумуляторға жүктемеленеді. Команда орындалып болғаннан кейін нәтиже қайта аккумуляторға жазылады, әрі бұл алынған мән келесі орындалатын операнд қолданатын мән болмаса онда оны негізгі жадыға сақтау керек. ЕМ аккумулятор базасындағы типтік архитектура келесі сурет 2.7 көрсетілген.

Аккумулятор базасындағы есептеу машинасының архитектурасы



Аккумулятор базасындағы есептеу машинасының архитектурасы

Аккумуляторға x жады ұяшығындағыларды жүктемелеу үшін load x командасы қолданылады. Осы команда арқылы ақпараттар x жады ұяшығынан оқылады, жадыдан шығу аккумулятор кірістеріне жалғасқан, әрі қарай оқылған деректер аккумуляторға жазылады.

Аккумулятордағы деректерді жадыға жазу үшін сақтау командағы store x қолданылады, оның орындалуы бойынша аккумялятор шығыстары шиналарға қосылады, одан кейін шинадағы информация жадыға жазылады. АЛҚ операцияларды орындау үшін жадыдағы операндтардың бірі деректер регистріне оқылады. Екінші операнд аккумуляторда болады. Деректер регистрінің және аакумулятордың сәйкес шығыстары АЛҚ қосылады. Жұмыс аяқталғаннан кейін АЛҚ нәтижелері аккумуляторға жазылады.

Аккумуляторлық КЖА артықшылықтары ретінде: қысқа командалар және командалардың декодталуының қарапайымдылығы алынады. Бірақ бір ғана регистрдің болуы жадыға қайта-қайта қатынауды туындатады.

Аккумуляторлық КЖА ертеректегі ЕМ әйгілі болды, мысалы, IBM 7090, DEC PDP-8.

Регистрлік архитектура

Берілген типтегі машиналарда процессордың құрамына жалпы міндетті регистрлар (ЖМР) кіреді. Регистрлердің разрядтылықтары әдетте машмна сөзімен сәйкес болады. Кез-келген регистрге оның нөмірін көрсетіп қатынауға болады. ЖАР саны CISC типіндегі архитектураларды әдетте көп емес (8-ден 32-дейін), ал RISC-архитектурасында ЖМР көптүрлілігі қолданылады (бірнеше жүз), бірақ бұл екі команда да тек екі, үш командаңың орындалуын қамтамасыздандыра алады.

Регистрлік архитектура операндтардың орналасуын келесі форматар бойынша орындалуына мүмкіндік береді:

- ⟨ регистр-регистр;
- ⟨ регистр-жады;
- ⟨ жады-жады.

Регистрлік архитектура

Форматтардың әрбірінің өзінің сәйкес артықшылықтары мен кемшіліктері бар (кесте 2.4).

(m, n) түріндегі өрнектерде, m – негізгі жадыда сақталатын операндтар саны, ал n — арифметикалық немесе логикалық өңдеулердің командаларындағы операндтардың жалпы саны.

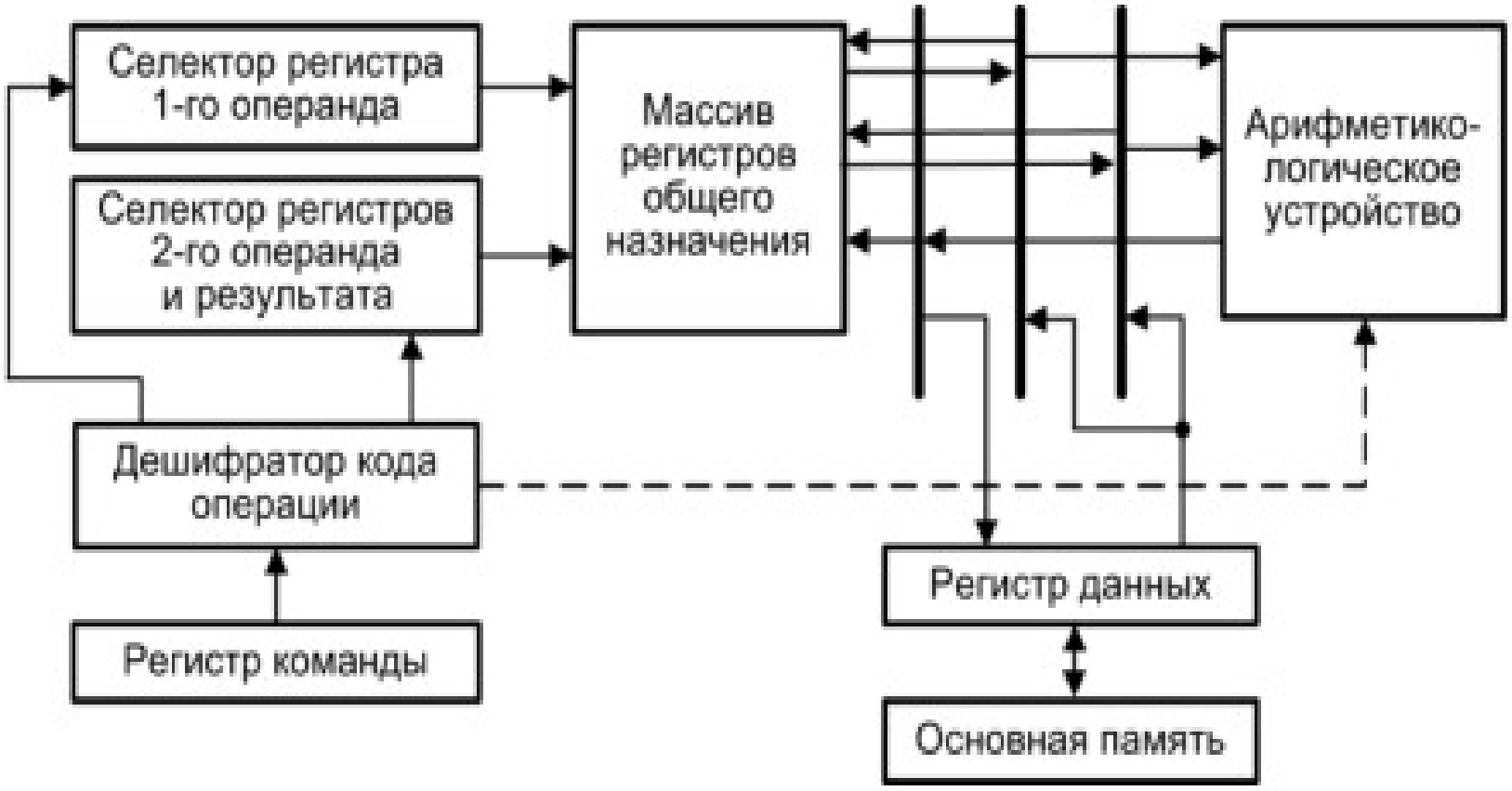
«регистр-регистр» форматы RISC типіндегі ЕМ, «регистр-жады» форматының командалары CISC-машиналары үшін, және соңғысы «жады-жады» форматы тиімді емес деп есептеледі, бірақ CISC класындағы қиын машиналарда қолданылады.

Операндтарды орналастыру варианттарының салыстырмалы бағасы

Вариант	Артықшылықтары	Кемшіліктері
Регистр-регистр (0, 3)	Простота реализации, фиксированная длина команд, простая модель формирования объектного кода при компиляции программ, возможность выполнения всех команд за одинаковое количество тактов	Большая длина объектного кода, из-за фиксированной длины команд часть разрядов в коротких командах не используется
Регистр-жады (1, 2)	Данные могут быть доступны без загрузки в регистры процессора, простота кодирования команд, объектный код получается достаточно компактным	Потеря одного из операндов при записи результата, длинное поле адреса памяти в коде команды сокращает место под номер регистра, что ограничивает общее число РОН. CPI зависит от места размещения операнда
Жады-жады (3, 3)	Компактность объектного кода, малая потребность в регистрах для хранения промежуточных данных	Разнообразие форматов команд и времени их исполнения, низкое быстродействие из-за обращения к памяти

ЖМР базасындағы ЕМ архитектурасы

Регистрлік архитектурадағы ЕМ мүмкін болатын ақпараттық тракті және құрылымының командалар жүйесі келесідегідей беріледі (сурет 2.8).



ЖМР базасындағы ЕМ архитектурасы

Регистрлерді жадыдан жүктемелеу операциялары және регситрдегі мәндерді жадыға сақтау аккумулятордағы операциялармен бірдей. Ерекшелігі тек керекті регистрді таңдау ғана.

АЛҚ операцияларды орындау келесілерден тұрады:

- бірінші операндтың орнын анықтау (регистр немесе жады);
- бірінші операндтың регистрінін таңдау немесе бірінші операндты жадыдан оқу;
- екінші операндтың орнын анықтау (регистр немесе жады);
- екінші операндтың регистрінін таңдау немесе екінші операндты жадыдан оқу;
- АЛҚ кірістеріне операндтарды беру және операцияны орындау;
- нәтиженің орнын анықтау (регистр немесе жады);
- нәтиже регистрін таңдау немесе жады ұяшығын әрі АЛҚ нәтижесін жазу.

ЖМР базасындағы ЕМ архитектурасы

Мұнда көңіл бөлетініміз, ол АЛҚ және регистрлік файлдар арасында үш шинаның болуы. Үш шинаның екеуі, ЖМР және АЛҚ арасында орналасқандары, ЖМР немесе жадыдағы деректерді АЛҚ беруді қамтамасыздандырады. Үшіншісі нәтижелерді олар үшін бөлінген регистрге немесе жады ұяшығына жазу үшін қолданылады.

Артықшылықтары: алынатын кодтың компактілігі, есептеудің жоғары жылдамдығы (жадыға қатынау жылдам регистрлерге қатынаумен ауыстырылады). Қазіргі кезде қолданылатын архитектуралар жүйесі болып табылады.

Жадыға бөлінген қатынау архитектурасы

Бұл архитектурады негізгі жадыға қатынау тек екі арнайы командалармен орындалады: load и store. Load/Store architecture деп атайды. load (жүктемелеу) командасы негізгі жадыдан мәндерді оқу және оны процессор регистріне орналастыруды қамтамасыздандырады. Ақпараттарды кері бағытта ауыстыру store (сақтау) командасымен орындалады. Барлық командалардағы ақпараттарды өңдеу операндтары тек процессор регистрлерінде ғана бола алады. Оперция нәтижесі де регистрлерге жазылады.

Бөлінген жадыға қатнауы бар ЕМ архитектурасы